UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Jhonatan Alves

UMA ABORDAGEM BASEADA EM PLANEJAMENTO NÃO-DETERMINÍSTICO E SAT PARA A COMPOSIÇÃO RESILIENTE E AUTOMÁTICA DE WEB SERVICES

Florianópolis

2016

Jhonatan Alves

UMA ABORDAGEM BASEADA EM PLANEJAMENTO NÃO-DETERMINÍSTICO E SAT PARA A COMPOSIÇÃO RESILIENTE E AUTOMÁTICA DE WEB SERVICES

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação para a obtenção do Grau de Mestre em Ciência da Computação.
Orientadora: Dr.ª Jerusa Marchi
Coorientador: Dr. Renato Fileto

Florianópolis

2016

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Alves, Jhonatan
UMA ABORDAGEM BASEADA EM PLANEJAMENTO NÃO DETERMINÍSTICO E SAT PARA A COMPOSIÇÃO RESILIENTE E AUTOMÁTICA DE WEB SERVICES / Jhonatan Alves ; orientadora, Jerusa Marchi ; coorientador, Renato Fileto. - Florianópolis, SC, 2016.
128 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da Computação. 2. Composição de Web services. 3. Planejamento Não-determinístico. 4. Satisfazibilidade Booleana. I. Marchi, Jerusa . II. Fileto, Renato. III. Universidade Federal de Santa Catarina. Programa de Pós Graduação em Ciência da Computação. IV. Título.

Jhonatan Alves

UMA ABORDAGEM BASEADA EM PLANEJAMENTO NÃO-DETERMINÍSTICO E SAT PARA A COMPOSIÇÃO RESILIENTE E AUTOMÁTICA DE WEB SERVICES

Esta dissertação foi julgada adequada para obtenção do título de mestre e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Florianópolis, 15 de dezembro de 2016.

_____
Profª. Carina Friedrich Dorneles, Drª.
Coordenadora do Programa

_____
Prof. Renato Fileto, Dr.
Universidade Federal de Santa Catarina
Coorientador

Banca Examinadora:

_____
Profª. Jerusa Marchi, Drª.
Universidade Federal de Santa Catarina
Orientadora

_____
Prof. Cláudio Cesar de Sá, Dr.
Universidade Estadual de Santa Catarina (videoconferência)

_____
Prof. Frank Augusto Siqueira, Dr.
Universidade Federal de Santa Catarina

_____
Prof. Elder Rizzon Santos, Dr.
Universidade Federal de Santa Catarina

Dedico este trabalho aos meus pais Leomar Argentil Alves e Marlei Angelita Borges Alves. Profundamente agradeço por acreditarem em mim, por apostarem na minha carreira, pelo constante apoio aos meus estudos. Agradeço pelo amor e carinho que me foram dados. Por nunca terem me negando ajuda, por sempre estarem presentes, pelas palavras de conforto e sabedoria. Grato sou por não medirem esforços ao me educarem. Por terem me ensinado os valores da vida, a aprender com os erros, o significado de perseverar, o sabor da recompensa, a estender a mão, a dizer te amo. Hoje sei que posso dizer que sou um ser humano mais completo. Sei que a pessoa que sou é o reflexo da dedicação, do afeto, do amor que vocês tiveram por mim. Eu os amo. Com carinho, Jhonatan.

AGRADECIMENTOS

Nesses anos de mestrado, de muito estudo e dedicação, muitas pessoas fizeram parte de minha vida e gostaria de lhes agradecer, com as mais sinceras palavras, por terem me acompanhado nessa jornada, por fazerem parte dessa conquista. Primeiramente, agradeço à minha família, em especial meus pais Leomar Argentil Alves e Marlei Angelita Borges Alves, minha irmã Jhenifer Alves, meu primo Carlos Azevedo e meu cunhado Luiz Hubner, pelo apoio e pela compreensão. Agradeço à professora Dr. Jerusa Marchi, minha orientadora, e, sobretudo, uma amiga, pela excelente orientação, pela ajuda na produção de artigos, pelas palavras de sabedoria, por ter se abdicado de vários momentos de descanso para me ajudar, pelas boas conversas e cafés. Grato sou pela coorientação do professor Dr. Renato Fileto que me ajudou no entendimento de vários conceitos relacionados ao tema dessa dissertação e pela ajuda na escrita de artigos. Quero dizer obrigado às novas amizades que fiz, por se envolverem em minha vida, pelo apoio e torcida. Em especial, grato sou à Yohana Hoffmann e Priscila Castilho. Gostaria de agradecer ao pessoal do laboratório de Inteligência Artificial e Tecnologia Educacional (IATE) no qual desenvolvi minhas pesquisas. Obrigado Allan Sabino, Darlan Anschau e Tiago Royer pelas conversas e boas risadas. Também gostaria de agradecer ao pessoal da secretaria da pós-graduação, em especial Katiana Castro e Manoella Torres, pelo excelente serviço prestado, torcida e amizade. Por fim, gostaria de agradecer aos professores Drs. membros da banca avaliadora, Cláudio Cesar de Sá, Elder Rizzon Santos e Frank Siqueira, por aceitarem o convite em avaliar o meu trabalho, pelo apoio e sugestões que contribuíram para um trabalho de qualidade.

Eu penso no tempo em que estarei tranquilo... Quando as vaidades e as paixões fúteis se consumirem, e os meus olhos, minhas mãos e minha mente poderão enternecer... Finalmente... E as canções serão suaves e flutuarão no ar...

Amiri Baraka

RESUMO

Web services têm sido cada vez mais adotados por organizações na implantação de processos de negócio e integração de sistemas heterogêneos. Todavia, os Web services estão vulneráveis a uma série de problemas de execução (serviços indisponíveis, resultados insatisfatórios, descumprimento de parâmetros de qualidade, violações de SLA, falhas de comunicação, dentre outros) que podem fazer com que os seus comportamentos sejam diferentes do esperado, impedindo-os de cumprirem com sucesso as tarefas para as quais foram designados. Neste sentido, é importante levar em consideração que o comportamento dos Web services é intrinsecamente não-determinístico, muitas vezes inesperado e inadequado. Vários trabalhos têm sido propostos para se obter, de modo eficiente, composições de Web services. Entretanto, com algumas exceções, a maioria ignora os problemas de execução que podem ocorrer em tempo de execução e afetar as composições. Neste contexto, este trabalho apresenta uma abordagem para a obtenção de composições resilientes de Web services, isto é, composições capazes de contornar problemas de execução para satisfazer os objetivos do usuário. A abordagem proposta combina planejamento não-determinístico e SAT (satisfazibilidade booleana) para se obter k planos alternativos (composições alternativas de Web services) que solucionam uma dada requisição, onde um plano é visto como uma sequência de ações, e as ações correspondem a invocações de operações de Web services. Os planos obtidos são fundidos em uma estratégia de contingência na forma de uma árvore de decisão binária (composição de Web services resiliente resultante). A estratégia de contingência permite a motores de execução de processos acompanhar o progresso da execução da composição e lidar com problemas que ocorrem em ambientes dinâmicos e não-determinísticos mediante a rápida seleção, com custo mínimo, de uma composição alternativa e compatível com aquela que falhou. Resultados de experimentos com a abordagem proposta mostraram que é possível obter estratégias de contingências em tempos relativamente baixos e com uma quantidade grande e satisfatória de planos. Em torno de um minuto foi possível construir árvores binárias com uma média de 6723 planos, e em torno de cinco minutos árvores binárias foram construídas com uma média de 15148 planos.

Palavras-chave: Composição Resiliente de Web Services Semânticos, Planejamento Não-determinístico, Satisfazibilidade Booleana.

ABSTRACT

Web services have been increasingly adopted by organizations to implement their business processes and to integrate heterogeneous systems. However, Web services are vulnerable to a wide number of execution problems (e.g., unavailable services, unappropriated results, noncompliance of quality parameters, service level agreement violations, communication failures, among others) which can make them behave differently from the expected and prevent them to comply with their tasks successfully. In this regard, it is important to take into account that the behavior of Web services is intrinsically nondeterministic, often unexpected and inappropriate. Several works have been proposed to efficiently obtain Web service compositions. However, with few exceptions, most of them disregard contingencies which may happen at runtime and affect the compositions. In this context, this paper presents an approach for obtaining resilient compositions of Web services, i.e, compositions that are able to circumvent problems which may occur in the execution environments in order to meet the user goals. This approach combines nondeterministic planning and SAT (boolean satisfiability) to obtain k plans (alternative compositions of Web services) that address a given user request, where a plan is a sequence of actions, and an action corresponds to an invocation of a Web services operation. The obtained plans are merged into a contingency strategy in the form of a binary decision tree (i.e., the resulting resilient composition of Web services). The contingency strategy enables process execution engines to keep track of the composition execution progress and deal with problems which happen in dynamic and nondeterministic environments by quickly selecting, with minimum cost, a suitable alternative composition to continue the execution towards the satisfaction of the user's goals. Experimental results using this approach have showed that it is possible to obtain contingency strategies in relatively low times and with a large and satisfactory amount of plans. In about one minute it was possible to build binary trees with an average of 6723 plans, and in about five minutes binary trees were built with an average of 15148 plans.

Keywords: Resilient Composition of Semantic Web Services, Nondeterministic Planning, Boolean Satisfiability.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

GPS	General Problem Solver
QA3	Question-answering program 3
SAT	Satisfazibilidade Booleana
PDDL	Planning Domain Definition Language
FNC	Forma Normal Conjuntiva
AEI	Axioma de Estado Inicial
AO	Axioma de Objetivos
AES	Axioma de Estados sucessores
APC	Axioma de Pré-condições
AEM	Axioma de Exclusão mútua
WSDL	Web services Description Language
UDDI	Universal Description, Discovery and Integration
SOAP	Simple Object Access Protocol
HTTP	Hyper Text Transport Protocol
WSDL-S	Web services Description Language with Semantics
OWL-S	OWL-based Service Ontology
WSMO	Web Services Modeling Ontology
AESC	Axioma de Estados Sucessores Contraídos
AOC	Axioma de Objetivos Contraído

LISTA DE SÍMBOLOS

t	Instante de tempo
Σ	Problema de planejamento
$\mathbb{P}$	Conjunto de símbolos proposicionais
p	Símbolo proposicional em $\mathbb{P}$
G	Conjunto de objetivos do agente de planejamento
S	Conjunto de estados do mundo
s	Estado do mundo em S
s_0	Estado inicial do mundo em S
A	Conjunto de ações do agente de planejamento
a	Ação em A
γ	Função de transição determinística entre estados do mundo
Σ	Instância de um problema de planejamento
S_g	Conjunto de estados de objetivos
s_g	Um estado de objetivos em S_g
G_Σ	Grafo de transição de estados
V_Σ	Conjunto de vértices do grafo de transição G_Σ
E_Σ	Conjunto de arestas do grafo de transição G_Σ
Args	Lista de argumentos de uma ação
Pre	Conjunto de pré-condições de uma ação
Add	Conjunto de efeitos positivos de uma ação
Del	Conjunto de efeitos negativos de uma ação
ρ	Plano de ações determinísticas
ψ	Fórmula lógica proposicional
C	Cláusula
σ	Modelo de uma fórmula
$\mathbb{D}_{ND}$	Domínio de um problema de planejamento não-determinístico
Σ_{ND}	Instância de um problema de planejamento não-determinístico
γ'	Função de transição não-determinística entre estados
A_{ND}	Conjunto de ações não-determinísticas
a_{ND}	Ação não-determinística em A_{ND}
$\mathcal{K}$	Custo de execução de uma ação
op	Uma operação de Web service semântico

I	Conjunto de parâmetros de entrada de uma operação
O	Conjunto de parâmetros de saída de uma operação
Pre'	Conjunto de pré-condições de uma operação
ND'	Conjunto de efeitos não-determinísticos de uma operação
$\mathcal{K}'$	Custo de execução de uma operação
r	Requisição de usuário
$\mathcal{C}$	Composição de Web services semânticos
W	Repositório de Web services semânticos
W'	Web services compatíveis com a requisição de usuário
R	Relações entre parâmetros de saída e entrada de uma operação

SUMÁRIO

1 INTRODUÇÃO

A inteligência, ainda que não definida precisamente, pode ser vista como um conjunto de habilidades mentais que permite a um indivíduo perceber o meio e sobre ele solucionar problemas complexos. Tais habilidades como, por exemplo, raciocinar segundo as regras da lógica, compreender o significado daquilo que foi percebido, aprender com a experiência, planejar o alcance de objetivos, entre outros, são inerentes a seres com comportamento inteligente (LEGG; HUTTER, 2007).

Planejamento é o processo abstrato e explícito que consiste, basicamente, em enumerar um conjunto finito de ações a fim de resolver um problema e alcançar um estado desejado com eficiência. Deste modo, antecipa-se como o mundo se modifica à medida que as ações são executadas e, assim, torna-se possível evitar situações indesejáveis, cortar custos, administrar tempo e recursos limitados, entre outros fatores. Em outras palavras, planejamento tem como fim satisfazer, da melhor forma possível, um conjunto de objetivos determinando, segundo a análise de um conjunto de critérios, o melhor plano (sequência de passos) que soluciona um certo problema (GHALLAB; NAU; TRAVERSO, 2004).

Em Inteligência Artificial, planejamento automático refere-se à área na qual estudam-se técnicas que permitam a agentes autônomos (mais especificamente chamados de *planejadores* ou *agentes de planejamento*) simularem a habilidade de planejar de modo inteligente e sintetizar planos através da análise das características intrínsecas dos problemas a serem resolvidos e objetivos a serem satisfeitos (RINTANEN, 2005). Genericamente, um problema de planejamento automático pode ser representado como um grafo de transição de estados, onde os vértices representam estados do mundo e as arestas representam transições decorrentes das execuções das ações. Um plano é um caminho no grafo que parte da raiz em direção a uma folha na qual os objetivos estão todos satisfeitos.

A área de planejamento automático é um reflexo das pesquisas da década de 50 cujo foco era o desenvolvimento de *solucionadores gerais de problemas*, a exemplo do GPS (*General Problem Solver*) (ERNST, 1979) e QA3 (*Question-answering program 3*) (GREEN, 1969). Naquela época, almejava-se que estes sistemas solucionassem qualquer tipo de problema. No entanto, este sonho mostrou-se inviável, principalmente pela dificuldade de se obter algoritmos suficientemente genéricos e robustos que resolvessem problemas dos mais diversos domínios e pela alta explosão combinatória de estados dos espaços de busca das solu-

ções (VERVAEKE; LILLICRAP; RICHARDS, 2012). Nas décadas seguintes, conhecidas como *Era Clássica do Planejamento Automático*, as pesquisas focaram-se no desenvolvimento de planejadores dependentes de domínios (solucionadores para problemas específicos) e restritos a certas classes de complexidade computacional (P e NP, por exemplo) (GEFFNER; BONET, 2013). Nesta época, buscava-se o desenvolvimento de técnicas eficientes de busca por planos, onde eram abstraídos fatores do mundo real como, por exemplo, ações com efeitos não-determinísticos ou durativos, agentes de planejamento com capacidade de observação parcial ou nula, dentre outros.

Entretanto, com os avanços nas pesquisas em planejamento o leque de problemas solucionáveis expandiu-se para os mais diferentes domínios de problemas e classes de complexidade, relaxando-se as restrições que eram aplicadas no processo de busca por planos desde os primórdios da área de planejamento automático. Deste modo, planejamento automático passou a se aproximar mais dos problemas do mundo real e vários deles passaram a ser solucionados por diferentes técnicas e modelos de planejamento. Dentre tais problemas estão: composição de Web services (LEMOS; DANIEL; BENATALLAH, 2015; PEER, 2005), controle de tráfego urbano (JIMOH; CHRPA; MCCLUSKEY, 2014), jogos digitais (COTE et al., 2012), planejamento de trajetória para robôs (QUINTERO et al., 2011), controle de veículos espaciais (ESTLIN et al., 2003), manufatura de peças (SANDERSON; MELLO; ZHANG, 1990), dentre outros.

1.1 FORMULAÇÃO DO PROBLEMA

Composição de Web services é o processo de formar serviços mais complexos e sofisticados a partir da conexão de invocações de Web services existentes a fim de satisfazer requisições de usuários que não possam ser cumpridas completamente por apenas um Web service. Composições de Web services podem ser obtidas de forma *manual*, *semi-automática* ou *automática* (RAO; SU, 2005).

Na composição manual, os Web services são selecionados e ligados manualmente por um programador. Entretanto, a numerosa quantidade de Web services disponíveis na Web torna esta abordagem custosa, sujeita a erros, demandando tempo e esforço computacional para a descoberta de serviços adequados às necessidades do usuário. A composição semi-automática é feita incrementalmente. A cada iteração, uma ferramenta de seleção sugere Web services para o programador

que, por sua vez, escolhe manualmente quais deles adicionar à composição. Entretanto, para o programador, esta abordagem tende a ser um pouco trabalhosa já que há a necessidade de que se conheça previamente as configurações e elementos de ligação (*bindings*) dos Web services. A falta de tal conhecimento pode resultar em uma composição mal elaborada e com erros. Na composição automática, os Web services são descobertos, combinados e ligados automaticamente por um agente inteligente. Esta abordagem traz uma série de benefícios como, por exemplo, redução da mão-de-obra para selecionar Web services de grandes repositórios, substituição automática de Web services que se tornam indisponíveis ou por Web services que melhor atendam as necessidades do usuário, recuperação de falhas, dentre outros (KARDARAS, 2012).

Não obstante, para que a composição automática de Web services seja viável, a adição de descrições semânticas às operações se torna imprescindível. Isso porque a descrição puramente sintática dá margem a más interpretações das funcionalidades e não fornece informações suficientes para descobrir e compor adequadamente Web services que realizem as tarefas desejadas (BARTALOS; BIELIKOVA, 2011).

Ademais, os Web services estão vulneráveis a problemas de execução. Isso ocorre, principalmente, por causa de problemas decorrentes da natureza dinâmica e não-determinística das redes de computadores (ou seja, problemas na infraestrutura de transmissão que podem ocasionar falhas de comunicação, servidores indisponíveis, etc), não cumprimento das pré-condições para que as operações sejam executadas, falhas de software e hardware, alterações nas interfaces dos serviços, passagem de parâmetros em número e tipos incorretos, descumprimento de parâmetros de qualidade, violações de SLA, dentre outros. Logo, os Web services podem retornar diferentes tipos de respostas indesejadas e não cumprir com sucesso as tarefas para as quais foram designados (GIACOMO; PATRIZI, 2009; TRAVERSO; PISTORE, 2004).

Sob estas circunstâncias, é importante levar em consideração, durante o processo de composição, que o comportamento dos Web services é intrinsecamente não-determinístico, muitas vezes inesperado e inadequado. É importante que composições alternativas sejam auferidas para que o motor de execução, ao detectar algum problema na invocação de algum Web service ou algum resultado diferente do desejado, possa selecionar a composição alternativa mais adequada e com custo de adaptação mínimo, aumentando as chances de que os objetivos da composição sejam alcançados. Neste sentido, o que se obtém são composições *resilientes*, isto é, composições que são capazes de contornar

problemas e fornecer seus serviços de modo contínuo frente à ocorrência de falhas em ambientes de execução dinâmicos e não-determinísticos (CABRI, 2014).

1.2 MOTIVAÇÃO

Vários trabalhos têm sido propostos para se obter de modo eficiente e automático composições de Web services. Muitos destes trabalhos são baseados em técnicas que fazem uso de verificações de modelos (*model checking*) (MI et al., 2016; ZHANG; LI, 2016), cadeias de Markov (CHRISTIAN; BOHARA, 2016; FILIERI et al., 2012), redes de Petri (DECHSUPA; VATANAWOOD; THONGTAK, 2016; MATEO et al., 2015), grafos (BHATTACHARYA et al., 2016; SILVA; MA; ZHANG, 2016), planejamento automático (MEHDI; ZAROUR, 2016; NIU et al., 2016), dentre outros.

Em particular, planejamento automático e composição de Web services, quando vistos como uma sequencia linear de invocações, são problemas relativamente próximos. Em essência, ambos problemas consistem em ordenar um conjunto de *ações* (ou *operações*) a fim de cumprir com um conjunto de objetivos. Neste sentido, a composição de Web services tratada como um problema de planejamento visa obter como resultado um plano, onde o plano corresponde a uma composição de Web services e as ações do plano correspondem a invocações de operações. O plano deve, então, ser traduzido para um processo executável mapeando as ações a suas respectivas operações, para que, em seguida, um motor execute o processo resultante. Este tipo de solução se mostra interessante e promissora, pois permite tirar proveito das técnicas e ferramentas obtidas nas pesquisas em planejamento (MARKOU; REFANIDIS, 2016).

Dentre os modelos existentes de planejamento automático, o que mais se adequa ao problema da composição de Web services, visto os comportamentos anteriormente citados, é o planejamento não-determinístico, onde as ações são formadas por conjuntos de efeitos incertos (tal conjunto pode ser utilizado para representar os efeitos incertos das operações dos Web services). Assim, não se pode inferir com exatidão qual estado do mundo será alcançado antes de uma ação ser executada, mas é possível identificá-lo após a execução da ação observando quais efeitos foram produzidos (o que implica em analisar os valores associados aos parâmetros de saída de uma operação em uma mensagem SOAP de resposta).

A perspectiva segundo este modelo de planejamento é a elaboração de um plano condicional (também chamado de política), ou seja, um plano que indica ao agente de planejamento qual ação executar a partir de cada estado do mundo alcançado não-deterministicamente. Entretanto, planejamento não-determinístico pertencente à classe de complexidade *EXP-completo* (LITTMAN, 1997) e encontrar uma solução para um problema de planejamento deste modelo é um processo geralmente árduo, principalmente em virtude da necessidade do agente de planejamento raciocinar sobre todos os possíveis caminhos que alcançam os seus objetivos frente ao não-determinismo da execução de suas ações.

Todavia, eficientes algoritmos de planejamento clássico (isto é, planejamento determinístico) e independentes de domínio têm sido desenvolvidos e podem ser utilizados para solucionar, de modo eficaz, problemas de planejamento não-determinístico. Para tal, o problema de planejamento não-determinístico deve ser mapeando a um problema de planejamento clássico e, sob esta ótica, deve-se fazer uso de técnicas ou ferramentas eficientes para buscar por um conjunto de planos alternativos determinísticos. Tais planos podem, então, ser sobrepostos em um plano condicional, na forma de uma árvore de decisão binária, que determina qual plano alternativo executar de acordo com os efeitos gerados pelas ações não-determinísticas.

Desta forma, o plano condicional pode ser visto como uma coleção de composições de Web services alternativas (onde cada composição equivale a um plano determinístico) indicando qual composição deve ser executada quando alguma operação de Web service, devido a um problema de execução, falhar em produzir os efeitos desejados.

Uma das técnicas de planejamento clássico mais bem sucedidas para a obtenção de planos ótimos (em termos de número mínimo de ações a executar linearmente) é o planejamento satisfazível (também chamado de planejamento SAT) (GIUNCHIGLIA; MARATEA, 2007). De acordo com esta técnica, problemas de planejamento clássico são traduzidos em tempo polinomial para fórmulas lógicas proposicionais e solucionadores SAT são utilizados para determinar a satisfazibilidade de tais fórmulas. Caso uma dada fórmula seja satisfazível, então, pelo menos um plano pode ser extraído daquela fórmula. O sucesso desta técnica se dá, principalmente, pelo avanço no desenvolvimento de poderosos e robustos solucionadores SAT que apresentam bons tempos de resposta para solucionar problemas de planejamento de diferentes domínios.

1.3 OBJETIVOS

Este trabalho tem como objetivo apresentar uma abordagem que combina planejamento não-determinístico e SAT para se obter automaticamente composições resilientes de Web services semânticos. Neste sentido, problemas de composição de Web services semânticos podem ser vistos como problemas de planejamento não-determinístico que, por sua vez, são mapeados a SAT. Através de SAT, se possível, um conjunto de planos alternativos determinísticos são obtidos e sobrepostos em um plano condicional.

Os objetivos específicos desta dissertação são:

1. Apresentar um modelo formal para mapear problemas de composição de Web services a problemas de planejamento não-determinístico.

2. Apresentar um modelo formal para mapear problemas de planejamento não-determinístico a SAT.

3. Utilizar um solucionador SAT completo para que sejam obtidos k planos alternativos determinísticos com comprimento máximo N.

4. Combinar os k planos alternativos determinísticos em uma árvore de decisão binária, segundo critérios de qualidade e compatibilidade, para se obter um plano condicional. Tal plano pode ser visto como uma estratégia de contingência que indica qual composição alternativa deve ser executada frente aos problemas de execução dos Web services da composição quando eles não produzem os efeitos desejados.

5. Apresentar um framework que especifica as fases e tarefas necessárias para que composições resilientes de Web services, considerando Web services com as características apresentadas neste trabalho, sejam obtidas.

6. Implementar um protótipo que realiza a fase de planejamento do framework proposto, testar o seu desempenho e a sua escalabilidade.

1.4 ORGANIZAÇÃO DA DISSERTAÇÃO

Este trabalho está organizado da seguinte forma: o capítulo 2 discute os trabalhos correlatos ao tema desta dissertação. O capítulo 3 apresenta os fundamentos teóricos básicos sobre os principais temas relacionados a esta dissertação, a saber planejamento automático e composição resiliente de Web services. O capítulo 4 apresenta um framework para a obtenção de composições resilientes de Web services semânticos via planejamento não-determinístico e SAT. O capítulo 5 aborda a implementação parcial do framework apresentado no capítulo 4, testes realizados e discussão sobre os resultados obtidos. Por fim, as conclusões deste trabalho e perspectivas sobre trabalhos futuros são apresentados no capítulo 6.

2 TRABALHOS CORRELATOS

Muitos trabalhos têm sido propostos para que composições de Web services sejam obtidas de modo eficiente. Todavia, com algumas exceções, a maioria destes trabalhos ignoram as contingências (i.e, falhas) que podem ocorrer em tempo de execução e invalidar as composições (KALDELI; LAZOVIK; AIELLO, 2016; MARKOU; REFANIDIS, 2016). Neste capítulo, trabalhos correlatos ao tema desta dissertação são descritos, onde parte deles faz uso de técnicas de planejamento automático. É possível notar que, basicamente, as propostas para lidar com as contingências propõem reexecutar a operação que falhou em obter os efeitos desejados, substituí-la por uma operação alternativa, replanejar o plano obtido em parte ou no todo.

2.1 FRWSC: A FRAMEWORK FOR ROBUST WEB SERVICE COMPOSITION

Kholy e Fatatry (2016) propuseram o framework *FRWSC* para o monitoramento da execução de composições de Web services em BPEL (*Business Process Execution Language*) a fim de prevenir e recuperar falhas. Para isso, o framework faz a análise do tráfego de mensagens SOAP que os Web services da composição trocam entre si, verifica se tais mensagens contém informações sobre erros de execução e, então, delega uma possível solução para corrigir o erro, caso exista alguma estratégia previamente estabelecida para ele, a fim de evitar futuras falhas ou recuperar o sistema de falhas que tenham ocorrido.

O framework FRWSC é constituído de dois módulos principais, *detector de falhas* e *fornecedor de soluções*. O módulo *detector de falhas* (i.e, detector de problemas de execução) intercepta as mensagens SOAP trocadas entres os Web services, analisa os seus conteúdos a fim de detectar erros explícitos (erros descritos no corpo da mensagem SOAP entre as tags *fault* como, por exemplo, problemas de conexão de rede, erros HTTP, entre outros) e implícitos (número incorreto de parâmetros de entrada, parâmetros de entrada com tipos diferentes daqueles esperados, parâmetros que assumem valores fora de seu domínio, entre outros).

Tal módulo faz uso de uma base de conhecimento que o auxilia a descobrir as causas dos erros e falhas que podem ocorrer caso os erros não sejam corrigidos (como ocorre quando os tipos dos parâmetros

passados a uma operação não combinam com os tipos esperados pela operação, se o erro não for corrigido uma falha irá ocorrer quando a operação for executada, por exemplo). Quando um erro é detectado, a execução do Web service que gerou a falha é interrompida para que o módulo *fornecedor de soluções* proponha uma solução para reparar o erro detectado.

O módulo *fornecedor de soluções* aplica três tipos de soluções: (a) *solução simples*, onde pede-se ao usuário para corrigir a mensagem SOAP de invocação a uma operação (quando o número de parâmetros, tipos e valores que podem assumir estão incorretos). As soluções (b) *intermediária* e (c) *complexa* são empregadas quando uma mensagem SOAP de resposta é do tipo falha (uma resposta SOAP pode ser do tipo *falha* ou *sucesso*). Na *solução intermediária*, tal operação é substituída por alguma outra operação equivalente. Na *solução complexa*, um curso de operações que produzam parâmetros de saída e efeitos equivalentes àquela operação que falhou é planejado. Esta última solução é aplicada quando não há operação alternativa para substituí-la. Os tempos de respostas também são monitorados. Caso alguma resposta SOAP exceda o tempo limite de espera previamente definido, então ocorre uma violação de qualidade da composição e ela tem sua execução finalizada.

Embora alguns tipos de falhas sejam prevenidas ou tratadas pelo framework FRWSC, a compatibilidade semântica entre os tipos dos parâmetros das operações não é verificada. A *solução simples* analisa os arquivos WSDL dos Web services para determinar se os tipos dos parâmetros nas mensagens SOAP combinam com aqueles das operações invocadas. Entretanto, a falta de uso de ontologias pode indicar que dois parâmetros que são semanticamente equivalentes não combinam. Ademais, os autores alegam que utilizam um planejador automático, na *solução complexa*, para buscar um curso de operações que substitua a operação que falhou, entretanto os autores não trazem detalhes sobre a construção de tal curso. E sem o uso de semântica restringe-se as possibilidades de que tal curso de operações seja encontrado.

2.2 COST-SENSITIVE PROBABILISTIC CONTINGENT PLANNING FOR WEB SERVICE COMPOSITION

Markou e Refanidis (2016) propuseram um framework para obter composições de Web services através de planejamento probabilístico levando em consideração a probabilidade das operações de Web services falharem em obter os efeitos desejados. Para isso, o problema de se obter uma composição de Web services é mapeado a um problema de planejamento probabilístico na linguagem (P)PDDL (YOUNES; LITTMAN, 2004) (extensão da linguagem PDDL com suporte a efeitos probabilísticos). Em seguida, o problema de planejamento probabilístico obtido é traduzido para um problema de planejamento clássico determinizando as ações em um processo similar ao trabalho desta dissertação, onde um Web Service com n conjuntos de efeitos não-determinísticos dá origem a n ações com efeitos determinísticos.

Mais precisamente, uma operação considerada com chances de falhar é mapeada a uma ação com dois conjuntos de efeitos não-determinísticos, onde um conjunto representa os efeitos desejados, e outro conjunto indica os efeitos indesejados. Uma operação que é considerada não ter chances de falhar é mapeada a uma ação apenas com um conjunto de efeitos determinísticos, isto é, um conjunto de efeitos desejados. A busca por planos é feita mediante extensão do algoritmo A^* (HART; NILSSON; RAPHAEL, 1968). O agente de planejamento busca por k planos para o problema de planejamento determinístico, e a cada plano é calculada a sua *utilidade esperada* que combina os custos de execução das ações que o compõem e as suas probabilidades de falhas em uma função probabilística. A utilidade esperada indica ao agente de planejamento quanto um plano tem chance de findar com sucesso frente as contingências que ocorrem em tempo de execução.

Os k planos obtidos são ordenados de forma decrescente segundo suas utilidades esperadas. O plano com a maior utilidade esperada é tomado como plano base sobre o qual os demais planos serão fundidos formando, assim, uma árvore de decisão binária como um plano condicional, onde os nós representam estados do mundo e as arestas representam ações a serem tomadas a partir de tais nós. Quando uma ação de um plano é considerada ter chances de falhar em gerar os efeitos desejados, então ao ser adicionada à árvore cria uma arestas à direita do nó corrente indicando que a ação respondeu com os efeitos desejados e uma aresta é adicionada à esquerda do nó corrente indicando que a ação falhou em obter os efeitos desejados. Caso a ação não tenha chances de falhar, então apenas uma aresta à direita do nodo corrente

é criada.

Para cada nodo alcançado à esquerda (nodo de falha) o plano com a maior utilidade esperada é escolhido para ser inserido na árvore e é removido da lista de planos obtidos. A construção do plano contingente finda quando não há mais planos a serem adicionados na árvore. Apesar de os autores considerarem que os possíveis efeitos dos serviços possuem probabilidade de ocorrência, os autores não apresentam nenhum estudo que atrele valores probabilísticos condizentes com a realidade, os valores são atribuídos de forma aleatória. Os autores consideram que nem todo Web service pode falhar, porém em uma abordagem mais real nenhum serviço está livre de falhar. Logo, por mais que mínima, sempre existe a possibilidade de que uma composição seja invalidada por uma falha. Ademais, em seus testes preliminares, os autores encontraram um máximo de 8 planos para construir árvores binárias.

2.3 FAILURE RECOVERY IN DISTRIBUTED MODEL COMPOSITION WITH INTELLIGENT ASSISTANCE

Huang et. al (2015) propuseram um framework que integra diferentes estratégias de recuperação de falhas: (i) *reexecutar*; (ii) *substituir*; (iii) *inserir*; e (iv) *reorganizar*. Na estratégia de reexecutar, tenta-se novamente executar a operação que falhou após um certo período de tempo previamente configurado. Na estratégia de substituir, a operação que falhou é substituída por uma sequência de operações que gera efeitos equivalentes aos seus evitando que o restante da composição seja replanejada. Neste caso, os parâmetros de entrada e saída da operação a ser substituída se tornam o estado inicial do mundo e objetivos a serem alcançados, respectivamente.

Em inserir, assim como em substituir, uma subsequência de operações é gerada e inserida entre duas operações quando ocorre incompatibilidade entre os seus parâmetros devido à modificações nas interfaces dos Web services a quais eles pertencem. Tal subsequência tem como objetivo gerar os parâmetros de entrada da operação invocada a partir dos parâmetros de saída da operação invocadora. E, finalmente, em reorganizar, a composição de Web services obtida é descartada e uma nova composição é gerada quando verifica-se que não é possível obter sucesso com as demais estratégias.

O framework é composto por dois módulos: *motor de execução* e *controlador de processos*. O motor de execução executa e monitora

uma composição, descrita como um processo na linguagem BPEL, interceptando as mensagens SOAP trocadas entre os Web services que a compõem a fim de detectar erros e tratar falhas. Se algum erro for detectado, então o controlador de processos é acionado para determinar uma estratégia para tratar uma certa falha. Para isso, tal módulo faz uso de uma ontologia que descreve diferentes tipos de erros relacionando-os às falhas e quais estratégias de recuperação são aplicáveis a elas. Caso seja inferido que a estratégia a ser empregada é substituir, inserir ou reorganizar, então o planejador MBP (*Model Based Planner*) (BERTOLI et al., 2001) é invocado para buscar por uma sequência de operações (plano) que será alternativa àquela sequência que se tornou inválida devido a uma certa falha.

Entretanto, os autores não consideram nenhum cálculo para determinar a compatibilidade semântica entre os parâmetros das operações para obter as composições de Web services, nem critérios como custo são considerados para obter uma composição com custo mínimo. A estratégia de reorganizar poderia ser evitada se diferentes composições fossem previamente obtidas, assim, caso o controlador de processos identificasse que as estratégias de substituir e inserir não podem ser empregadas na composição corrente, então uma composição alternativa e compatível com aquela que se tornou inválida poderia ser executada.

2.4 A RESILIENT FRAMEWORK FOR FAULT HANDLING IN WEB SERVICE ORIENTED SYSTEMS

Wang, W. et. al (2015) propuseram um framework para obter composições de Web services capazes de lidar com falhas provenientes de *erros de rede*, *incompatibilidade entre os tipos dos parâmetros de operações* e *violações de parâmetros de qualidade* e *violações em SLAs*. Para isso, o framework é composto por três módulos: o *analisador de exceções*, *decisor* e *selecionador de estratégia*. O analisador de exceções é responsável por analisar o arquivo de log gerado pelo motor de execução de processos a fim de extrair informações sobre o comportamento dos Web services determinando quando falhas os impedem de executar com sucesso. O módulo decisor determina quais estratégias aplicar a fim de recuperar a composição das falhas detectadas pelo módulo anterior permitindo à composição alcançar os objetivos do usuário.

As estratégias adotadas são as seguintes: (i) *substituir* a operação que viola um atributo de qualidade ou SLA por outra operação equivalente; (ii) *reinvocar* a operação que falhou em casos onde os ser-

viços se encontram indisponíveis, ou quando ocorrem exceções devido a incompatibilidade entre parâmetros; (iii) reverter (rollback) os efeitos produzidos por operações afetadas por falhas de conexão de rede.

Entretanto, os autores não consideram que a estratégia de substituir nem sempre pode ser aplicada, pois, de fato, uma operação equivalente àquela que falhou nem sempre existe. Além do mais, os autores também não consideram usar compatibilidade semântica para determinar quando uma dada operação é equivalente à outra. Logo, restringe-se as chances de que uma operação alternativa seja encontrada. Ademais, reinvocar pode não ser uma boa estratégia caso um dado serviço demore muito tempo para voltar a estar disponível ou caso este não se torne mais disponível. Composições alternativas poderiam ser auferidas para complementar as estratégias de substituir e reinvocar quando ambas falham.

2.5 SELF-ADAPTIVE RESILIENT SERVICE COMPOSITION

Torrez e Holvoet (2014) propuseram um sistema que obtém composições que lidam com a indisponibilidade dos Web services. Para isso, a abordagem adotada é baseada em um mecanismo de coordenação de agentes conhecido como *delegateMAS*. Este mecanismo se adequa bem a sistemas distribuídos em larga escala e aplicações de controle que precisam coordenar recursos, tais como gerenciamento de tráfego e logística. O sistema desenvolvido recebe um modelo de uma composição a ser obtida como um workflow de tarefas que satisfazem os objetivos do usuário. O mecanismo *delegateMAS* busca por um conjunto de operações de Web services que estão distribuídos em uma rede de serviços em larga escala a fim de identificar operações que possam realizar as tarefas do workflow.

O mecanismo *delegateMAS* é constituído por um conjunto de agentes, cada um com um papel específico. O agente *TaskAgents* monitora a composição a fim de determinar se algum Web service se torna indisponível. Caso isso ocorra, o agente TaskAgent delega aos agentes *IntentionAnts* a tarefa de distribuir sobre as redes de serviços informações (denominadas de ferormônios) relativas à operação que falhou para que uma operação compatível seja encontrada. As informações abrangem especificações sobre parâmetros de qualidade, como custo de execução, tempo médio de resposta, disponibilidade, dentre outros.

O ferormônio percorre cada nodo, que por sua vez, é constituído por um agente do tipo *ResourceAgent* que detém todas as informações

relativas aos atributos do Web service residente naquele mesmo nodo. O agente ResourceAgent armazena o ferormônio caso o Web service a ele associado possua operação candidata a substituir aquela que falhou. O ferormônio espalhado pelas redes é usado pelos agentes do tipo *ExplorationAnts* para auxiliá-los a encontrar e listar os Web services candidatos. A heurística *Ant Colony Optimization* é utilizada para determinar qual caminho a ser percorrido por um agente *ExplorationAnts* evitando que ele percorra as redes exaustivamente. Tal heurística leva em consideração a concentração de ferormônio em um determinado caminho e a qualidade dos parâmetros desejados dos Web services naquele caminho.

Os agentes *ExplorationAnts* realizam suas buscas até que seus ciclos de vida expirem ou que tenham alcançado uma certa profundidade das redes. O agente TaskAgent escolhe o Web service cuja operação alternativa maximiza suas necessidades. Embora esta abordagem seja interessante o único tipo de falha tratada é aquela relacionada a indisponibilidade dos serviços.

2.6 FACING UNCERTAINTY IN WEB SERVICE COMPOSITIONS

Alferez e Pelechano (2013) propuseram um framework para que certos requisitos de uma composição (i.e, segurança, desempenho, custo, etc), previamente definidos pelo usuário, sejam preservados enquanto a composição de Web services executa. O framework proposto monitora a execução da composição a fim de identificar falhas que possam violar os requisitos, ou seja, quando os seus valores se tornam inaceitáveis (como quando o custo de execução de uma operação se torna mais alto do que o esperado, por exemplo).

Três módulos compõem o framework: *planejador de evolução*, *motor de reconfiguração* e *motor de execução*. Ademais, os modelos (workflows de tarefas especificados na linguagem *Business Process Model and Notation* - BPMN) da composição e das estratégias são salvos em uma base para que possam ser utilizados futuramente quando necessários. O módulo *planejador de evolução* monitora a execução dos Web services a fim de determinar quando uma falha afeta negativamente uma operação violando um certo requisito. Se for o caso, o módulo ao analisar uma ontologia que determina a estratégia a ser aplicada para preserva o requisito. Então, o módulo *motor de reconfiguração* funde o modelo da composição com o modelo (também um workflow de tarefas em BPMN) da estratégia selecionada.

A fusão é feita mediante criação de uma relação de paralelismo entre a tarefa do modelo que é realizada pela operação afetada pela falha e o modelo da estratégia escolhida. Como resultado, quando tal operação for executada a estratégia também o será. O modelo da composição é enriquecido com novas estratégias conforme necessário, assim se obtém ao longo do tempo um modelo abstrato da composição evoluído que indica quais são as mudanças necessárias que a composição física de Web services (composição na qual as tarefas estão substituídas por operações) deve atender para preservar os seus requisitos afetados.

A composição física enriquecida é, então, traduzia a um código em BPEL4WS que, por sua vez, é adicionado ao diretório de *deploy* do motor de execução que cria e executa um processo para o novo código. Para cada enriquecimento do modelo abstrato um novo processo é criado e executado. O motor executa diferentes processos ao longo do tempo conforme é verificado que o último processo criado não é capaz de preservar os requisitos.

Embora o framework proposto evite que composições de Web services se tornem inválidas preservando alguns tipos de requisitos através da análise do impacto negativo que falhas possam incidir sobre as operações, composições alternativas não são auferidas para os casos onde não é possível alcançar os objetivos do usuário quando os efeitos desejáveis não são possíveis de serem produzidos.

2.7 DYNAMIC PLANNING APPROACH TO AUTOMATED WEB SERVICE COMPOSITION

Kuzu e Cicekli (2012) propuseram um framework para a composição automática de Web services que intercala fases de planejamento e execução das operações de Web services. Para isso, o problema de se obter uma composição de Web services é descrita na linguagem PDDL e o planejador *Simplanner* (ONAINDIA et al., 2001) é empregado para obter de modo incremental um plano através de uma busca em profundidade no espaço de estados. A cada passo, o agente retorna a melhor ação a ser executada, segundo critérios previamente definidos, e a repassa ao módulo de execução que invoca o Web Service com operação corresponde à ação. Enquanto a operação é executada, o agente busca pela próxima melhor ação.

A intercalação se mantém até que seja gerada uma sequência de invocação de operações que satisfaçam a requisição do usuário ou que seja determinado que uma solução não existe. A execução dos Web

services é monitorada e caso alguma falha ocorra o *tratador de eventos inesperados* é acionado. Dois tipos de falhas são consideradas: *Web services indisponíveis* e *informações desconhecidas*. No primeiro caso, o agente busca por um caminho alternativo no espaço de busca que produza os mesmos parâmetros de saída da operação que falhou a fim de substituí-la. Se tal caminho não existir, então a busca é encerrada e assume-se que não há solução para o problema. No segundo caso, o usuário é questionado sobre o valor de um determinado parâmetro necessário para que uma certa operação seja invocada quando não é de conhecimento do agente tal valor. Se o usuário não puder informá-lo, então o agente pode, como no primeiro caso, determinar um caminho alternativo que produza os mesmos parâmetros de saída e que não necessite do parâmetro desconhecido que foi questionado sobre o seu valor ao usuário.

Apesar de o planejador Simplanner apresentar bons tempos de busca para encontrar composições de Web services, ele apresenta duas grandes deficiências: ao buscar por um plano, o agente pode ficar preso em um estado morto (*dead-end*) ou ficar preso em um loop (sequência de ações). O espaço de busca não é completamente representado, logo a ação escolhida a cada fase de planejamento é a melhor ação local e, assim, não há garantia de que o plano ótimo possa ser encontrado.

2.8 FAST DYNAMIC RE-PLANNING OF COMPOSITE OWL-S SERVICES

Klusch e Renner (2006) propuseram um framework, o *OWLS-XPlan+*, para compor Web services permitindo que a composição obtida seja replanejada, em tempo de execução, caso alguma falha a impeça de satisfazer os objetivos do usuário. O framework OWLS-XPlan+ é constituído pelos módulos: *pré-processamento* e *planejamento*. O módulo de *pré-processamento* é responsável por mapear o problema de se obter uma composição de Web services a um problema de planejamento descrito na linguagem de planejamento PDDL.

O módulo de *planejamento* faz uso do planejador *FF planer* (HOFFMANN; NEBEL, 2001) que obtém um plano relaxado para cada objetivo do agente, onde tal plano é aquele no qual os efeitos negativos das ações são ignorados. Deste modo, pode-se estimar rapidamente quais são as ações mais prováveis (chamadas de ações úteis) a partir delas, originar planos completos, ou seja, planos onde as ações são formadas por seus respectivos conjuntos de efeitos positivos e negativos. Heuristicamente,

o agente decide quais das ações úteis são as melhores para iniciar uma nova busca por um plano completo. Com o foco do agente nas ações úteis, o espaço de busca pode ser consideravelmente reduzido.

Ao módulo de planejamento está agregado um *event listener* para detectar os seguintes eventos em tempo de execução: (i) *Web services indisponíveis*; (ii) *Web services que se tornam disponíveis*; e (iii) *mudança de objetivos*. No caso do primeiro evento, o agente busca por um Web service com operação alternativa que substitua a operação do Web service que se tornou indisponível. Caso tal operação não tenha sido encontrada, então o agente replaneja a composição a partir do ponto de falha, isto é, a sequência de ações do plano que originou a composição é mantida a partir do seu início até a ação imediatamente anterior àquela que falhou para buscar uma subsequência de ações a partir do ponto de falha que satisfaça os objetivos do usuário. No caso do segundo evento, o agente verifica se há a possibilidade de gerar um plano menor em relação ao plano corrente. Isso é feito calculando o ponto no plano onde uma das operações do Web service pode ser inserida, e o restante do plano é replanejado. Se for possível tal inserção e se o novo plano for menor, então o plano corrente é substituído pelo novo plano.

Caso os objetivos do usuário mudem, o agente verifica se há possibilidade de que alguma parte do plano corrente possa ser aproveitada no novo plano a ser calculado. Se for possível, o agente determina o ponto no plano corrente a partir do qual a nova parte deve ser replanejada. Se não for possível, o agente descarta o plano corrente e busca por um novo plano do início. Nos três casos, o estado inicial, sobre o qual a parte a ser replanejada é calculada, é composto pelos efeitos gerados pelas ações até o ponto de falha.

Embora o framework OWLS-XPlan+ tenha como princípio reutilizar o máximo de operações até o ponto de falha, a técnica baseada em planos relaxados direciona o agente a um parte do espaço de busca ignorando as demais regiões. Neste sentido, não existe garantia de que o plano ótimo possa ser encontrado, mas apenas um plano que satisfaz os objetivos do usuário sem critério de qualidade. Ademais, não considera-se compatibilidade semântica entre os parâmetros das operações, nem mecanismos de compensação para desfazer os efeitos gerados pelas operações no caso da composição ter que ser replanejada desde o início.

No próximo capítulo serão abordados fundamentos teóricos básicos sobre os principais temas relacionados a esta dissertação, a saber planejamento automático e composição resiliente de Web services.

3 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os fundamentos teóricos básicos sobre os principais temas relacionados a esta dissertação. Inicialmente, fala-se de planejamento automático clássico, satisfazível e não-determinístico. Ao final deste capítulo, conceitos sobre composição de Web services semânticos e resiliência são descritos.

3.1 PLANEJAMENTO AUTOMÁTICO

Em Inteligência Artificial, planejamento automático refere-se à área que se preocupa com a representação do conhecimento e com o desenvolvimento de mecanismos de manipulação eficientes, que possibilitem a um agente autônomo (também chamado de agente de planejamento) raciocinar e decidir qual ação executar a cada instante de tempo de modo que este cumpra com os seus objetivos. Caso o agente alcance seus objetivos, o conjunto de ações tomadas para tal fim é chamado de *plano*. Caso contrário, diz-se que não é possível decidir se há solução para o problema de planejamento em tempo tratável (GHALLAB; NAU; TRAVERSO, 2004).

Para planejar, o agente necessita da representação do domínio de um problema de planejamento e de uma instância sobre o domínio, descritos conforme alguma linguagem para representação do conhecimento como, por exemplo, PDDL (*Planning Domain Definition Language*) (MCDERMOTT et al., 1998). Basicamente, estas linguagens descrevem o domínio como sendo constituído de um conjunto de objetos que formam o mundo onde o agente planeja, predicados que estabelecem os estados de tais objetos e ações executáveis pelo agente, ao passo que a instância de planejamento sobre o domínio é constituída por um estado inicial, no qual inicia-se a busca por um plano, e por um conjunto de objetivos que o agente deve alcançar.

O agente busca por um plano seguindo algum modelo de planejamento, onde o modelo reflete o mundo sobre o qual o agente age. Por exemplo, o modelo de planejamento clássico prevê um mundo estático e determinístico, ao passo que um dos modelos de planejamento não-determinístico prevê um mundo dinâmico e ações com conjunto de efeitos incertos. O plano encontrado será executado sobre o mundo que reflete as características do modelo de planejamento adotado (GHALLAB; NAU; TRAVERSO, 2004).

A figura 1 ilustra os componentes básicos envolvidos em um processo de busca por um plano.

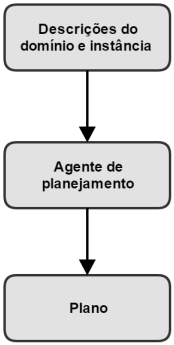

Figura 1 – Componentes básicos de um problema do planejamento.

Dentre os modelos de planejamento o mais adotado é o modelo do *planejamento clássico*, considerado o mais simples, onde assume-se um conjunto de restrições sobre o domínio dos problemas de planejamento. Tais restrições implicam que:

- As ações possuem efeitos determinísticos. Deste modo, o agente sabe com exatidão o estado que será alcançado após a aplicação de uma ação.
- Os efeitos das ações são instantâneos. Se uma ação é aplicada em um instante de tempo t seus efeitos findam no instante de tempo $t+1$.
- O custo do agente transitar de um estado para outro é nulo.
- Os estados do mundo são totalmente observáveis. Logo, o agente conhece com exatidão todos os fatos que compõem um determinado estado.
- O agente planeja em modo *offline*, ou seja, eventos exógenos são ignorados no processo de busca. As mudanças que ocorrem no mundo são unicamente causadas pelas execuções do agente.
- Um plano é uma sequência finita linearmente ordenada de ações.
- Não há suporte para objetivos estendidos, isto é, sub-metas, preferências, entre outros.

Os formalismos a seguir são baseados em Ghallab, M. et. al (2004).

Definição 3.1. *O domínio de um problema de planejamento clássico é definido como sendo uma 4-tupla* $\mathbb{D} = \langle \mathbb{P}, S, A, \gamma \rangle$*, tal que:*

$\mathbb{P} = \{p_1, p_2, \ldots, p_n\}$ *denota um conjunto de símbolos proposicionais, onde* $p \in \mathbb{P}$ *é um símbolo que identifica um fato do mundo (i.e, uma proposição).*

$S = \{s_1, s_2, \ldots, s_m\}$ *denota um conjunto de estados, onde* $S \subseteq 2^{\mathbb{P}}$*, e cada estado* $s \in S$ *é um conjunto sobre* $\mathbb{P}$*.*

$A = \{a_1, a_2, \ldots, a_z\}$ *denota um conjunto de ações e* $A(s)$ *refere-se ao conjunto de ações aplicáveis no estado* $s \in S$*.*

$\gamma : S \times A \to S$ *denota uma função de transição determinística entre os estados de* S*.*

Definição 3.2. *Uma instância de um problema de planejamento clássico sobre um domínio* $\mathbb{D}$ *é uma tupla* $\Sigma = \langle s_0, G \rangle$*, tal que:*

$s_0 \in S$ *denota o estado inicial do mundo sobre o qual o agente inicia a sua busca por um plano.*

$G \subseteq \mathbb{P}$ *denota um conjunto de objetivos do agente.*

Seja $S_g = \{s_1, s_2, \ldots, s_h\}$ um conjunto de estados de objetivos (estados nos quais os objetivos estão satisfeitos), tal que, $S_g \subseteq S$ e $\forall s \in S_g : G \subseteq s$. Para uma instância onde $|S_g| = k$, com $k > 0$, existem pelo menos k planos que a solucionam. Se $k = 0$, então não existe solução para o problema.

Esquematicamente, o espaço de busca para problemas de planejamento pode ser representado por um *grafo de transição de estados* $G_\Sigma = (V_\Sigma, E_\Sigma)$, onde os vértices V_Σ e as arestas E_Σ representam, nesta ordem, os estados e as ações em Σ. Se $\gamma(s, a) = s'$, com s, $s' \in S$ e $a \in A$, então há uma transição do estado s para o estado s' pela aplicação da ação a.

Definição 3.3. *Um estado* $s \in S$ *reflete um conjunto de fatos que compõem o mundo em um dado instante de tempo, tal que:*

$$s = \bigwedge p_i : p_i \in \mathbb{P}, 1 < i \leq j$$

Onde j refere-se ao número de símbolos proposicionais que compõem $s \in S$. Mais especificamente, os fatos descrevem os estados dos objetos sobre os quais as ações operam. Por exemplo, `livre(A)` e `sobre(A B)` descrevem que nenhum bloco está sobre o bloco `A` (fato 1) e que o bloco `A` está sobre o bloco `B` formando uma pilha (fato 2), respectivamente.

Definição 3.4. *Uma ação $a \in A$ é uma 4-tupla tal que $a = \langle Nome, Args, Pre, Add, Del \rangle$, onde:*

$Nome$ denota o nome da ação a.

$Args = \{x_1, x_2, \ldots, x_n\}$ denota uma lista de parâmetros, onde um parâmetro $x \in Args(a)$ refere-se a um objeto do mundo sobre o qual a ação irá operar.

$Pre = \{p_1, p_2, \ldots, p_m\}$ denota um conjunto de pré-condições que devem ser satisfeitas para que a ação a seja aplicável em um estado $s \in S$ do mundo.

$Add = \{p_1, p_2, \ldots, p_k\}$ denota um conjunto de efeitos positivos, ou seja, novos fatos que comporão o estado mundo $s' \in S$ alcançado após a execução da ação a.

$Del = \{p_1, p_2, \ldots, p_l\}$ denota um conjunto de efeitos negativos, ou seja, fatos que, fazendo parte do estado $s \in S$ no qual a ação a se aplica, não farão parte do estado $s' \in S$ alcançado.

Os componentes de uma ação $a \in A$ serão identificados, ao longo deste texto, por $Pre(a)$, $Add(a)$ e $Del(a)$. Ademais, os elementos que compõem tais conjuntos são, também, representados por símbolos proposicionais, tal que, $Pre(a)$, $Add(a)$, $Del(a) \subseteq \mathbb{P}$. Ao ser executada sobre um estado $s \in S$ do mundo, a ação a modifica-o eliminando e adicionando novos fatos. Como resultado, o mundo transita do estado s para um estado s'.

Definição 3.5. *O estado $s' \in S$, alcançado pela execução de uma ação $a \in A$ em um estado $s \in S$, é definido da seguinte forma:*

$$s' = (s - Del(a)) \cup Add(a)$$

Os fatos pertencentes ao estado s que não fazem parte do conjunto $Del(a)$ são transportados para o estado s'.

Definição 3.6. *A solução para um problema de planejamento* Σ *é um plano* $\rho = (a_1, a_2, \ldots, a_n)$*, ou seja, uma sequência linearmente ordenada de ações, com* a_1*,* a_2*,...,* $a_n \in A$*.*

A sequência de ações do plano ρ corresponde a uma sequência de transições de estados em G_Σ, de modo que $s_1 = \gamma(s_0, a_1)$, $s_2 = \gamma(s_1, a_2)$,..., $s_g = \gamma(s_{n-1}, a_n)$, para algum $s_g \in S_g$, se e somente se $\gamma(s_i, a_j) \neq \emptyset$.

Na próxima seção será apresentada a linguagem de planejamento padrão para a representação de problemas de planejamento, a linguagem PDDL. Ao final da seção, um exemplo demonstrará o uso da linguagem PDDL e os conceitos de planejamento clássico apresentados até então.

3.1.1 A Linguagem PDDL

PDDL (*Planning Domain Definition Language*) (MCDERMOTT et al., 1998) é a linguagem padrão para a representação de conhecimento em problemas de planejamento. A descrição de um problema é dividida em duas partes distintas: *domínio*, no qual são especificados os tipos de objetos conhecidos no mundo, predicados que descrevem genericamente os estados dos objetos, operadores que definem formatos padrão para as ações; e a instância sobre o domínio, que especifica os objetivos que o agente deve satisfazer, estado inicial sobre o qual a busca de um plano se inicia e os objetos manipuláveis pelo agente.

O domínio pode ser descrito conforme a seguinte gramática[1]:

Em PDDL não é obrigatório que os tipos dos objetos sejam definidos, entretanto na versão deste trabalho a definição dos tipos é mandatória pois, como será visto no capítulo 4, os tipos serão usados para especificar para as ações do agente sobre quais tipos de parâmetros de operações de Web services as ações irão executar. Em PDDL não existe conjunto de efeitos negativos (*Del*), a exclusão de uma proposição é feita mediante o uso do token `not` como em `(not (sobre(A B))`, por exemplo. É permitido definir hierarquicamente os tipos dos objetos, tais como, `inteiro racional - real`, onde `inteiro` e `racional` são subtipos de `real`. O símbolo `;` é usado para escrever comentários. Um problema de planejamento determinístico onde os parâmetros possuem tipos pode

[1] A versão da linguagem PDDL apresentada neste trabalho é um subconjunto daquela presente em (KOVACS, 2011). Esta versão restringe-se unicamente aos elementos essenciais ao mapeamento de problemas de composição de Web services a problemas de planejamento.

```
<domain>              ::=  (define (domain <name>)
                           <objects-types-def>
                           <predicates-def>
                           <action-def>+)
<objects-types-def>   ::=  (:types <types-list>)
<types-list>          ::=  <name>+ (- <type><types-list>?)?
<predicates-def>      ::=  (:predicates <predicates-list>)
<predicates-list>     ::=  (<predicate><parameters-list>)<predicates-list>*
<parameters-list>     ::=  <variable> - <type> <parameters-list>*
<predicate>           ::=  <name>
<type>                ::=  <name>
<variable>            ::=  ?<name>
<name>                ::=  <simbol><name>*

<simbol>              ::=  <letter>|<number>|<special-character>
<letter>              ::=  [a-z,A-Z]
<number>              ::=  [0-9]<number>*
<special-character>   ::=  [.|=|>|≥|<|≤|]
<action-def>          ::=  (:action <name>
                           :parameters (<parameters-list>)
                           <action-def-body>)
<action-def-body>     ::=  :precondition <preconditions>
                           :effect <effects>
<preconditions>       ::=  (and <predicates-list>)
<effects>             ::=  (and <predicates-list>
                           (not <predicate-list>)?)
```

ser descrito conforme a seguinte gramática:

```
<problem>         ::=  (define (problem <name>)
                       (:domain <name>)
                       <objects>
                       <init-state>
                       <goals>)
<objects>         ::=  (:objects <objects-list>+)
<objects-list>    ::=  <name> - <type>
<init-state>      ::=  (:init <prop-list> <functions>)
<prop-list>       ::=  (<proposition> <parameters-list>)+
<goals>           ::=  (goal:(and <prop-list> (not <prop-list>)*))
<proposition>     ::=  <name>
```

Exemplo 1. *Considere o problema do Mundo das Células, onde um robô deve encontrar uma rota entre um conjunto de células adjacentes partindo de uma posição inicial para uma posição final. Para o domínio deste problema foram definidos o tipo* `célula` *para objetos, os predicados* `sobre` *e* `adjacente` *que indicam, respectivamente, sobre qual* `célula` *o robô se encontra e quando dois objetos* `célula` *são adjacentes, a ação* `Mover` *que indica que o robô se move de uma* `célula ?x` *para uma* `célula ?y`*, desde que estes objetos sejam adjacentes. A descrição do domínio do Mundo das Células segundo a gramática da linguagem PDDL apresentada anteriormente é a seguinte*[2]*:*

```
(define (domain MUNDO_DAS_CÉLULAS)
(:types célula)
(:predicates (sobre ?x - célula)
             (adjacente ?y - célula ?z - célula))
(:action Mover
:parameters (?x - célula ?y - célula)
:precondition (and (sobre ?x))
:effect (and (sobre ?y)
        (not (sobre ?x)))))
```

Sobre o domínio do Mundo das Células o problema onde um robô, inicialmente sobre a `célula A`, deve se deslocar para a `célula C` descrito segundo a gramática para problemas de planejamento apresentada anteriormente é a seguinte:

```
(define (problem 3_CÉLULAS)
(:domain MUNDO_DAS_CÉLULAS)
(:objects (A - célula) (B - célula) (C - célula))
(:init (sobre A)
(adjacente A B) (adjacente B A)
(adjacente B C) (adjacente C B) )
(:goal (and (sobre C))))
```

A figura 2 ilustra o problema do Mundo das Células do exemplo 1. O ponto em vermelho representa o robô.

[2]Um exemplo que ilustra um problema de obter uma composição de Web services como um problema de planejamento será apresentado na seção 4.4, pois há a necessidade de que conceitos fundamentais ao tema desta dissertação ainda não discutidos sejam primeiramente apresentados.

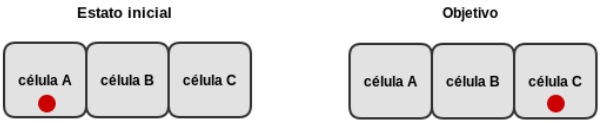

Figura 2 – Problema sobre o domínio do Mundo das Células.

Para este problema, os elementos da definição 3.2 (7-tupla de planejamento) são os seguintes:

- Instanciando os predicados `sobre` e `adjacente` com os objetos do tipo `célula` declarados na descrição do problema, obtém-se o conjunto $\mathbb{P} = \{$`(sobre A)`,`(sobre B)`,`(sobre C)`, `(adjacente A B)`, `(adjacente B A)`,`(adjacente B C)`,`(adjacente C B)`$\}$
- Através da extração das informações em `init` é possível obter o estado inicial do mundo $s_0 = \{$`(sobre A)`$\}$
- O conjunto de ações executáveis pelo agente é $A = \{$`Mover(A B)`, `Mover(B A)`, `Mover(B C)`, `Mover(C B)`$\}$
- Tendo em vista que inicialmente apenas o estado $s_0 \in S$ é conhecido, os demais estados do mundo são obtidos iterativamente segundo a função de transição γ que indica quais são os novos estados alcançados pela aplicação das ações, tal que:
  $\gamma : s_0 \times$ `Mover(A B)` $\rightarrow s_1$
  $s_1 \times$ `Mover(B A)` $\rightarrow s_0$
  $s_1 \times$ `Mover(B C)` $\rightarrow s_2$
  $s_2 \times$ `Mover(C B)` $\rightarrow s_1$
- O conjunto de estados do mundo é $S = \{s_0,s_1,s_2\}$ com $s_0 =$ $\{$`(sobre A)`$\}$, $s_1 = \{$`(sobre B)`$\}$, $s_2 = \{$`(sobre C)`$\}$
- Através da extração das informações em `goal` é possível obter o objetivo $G = \{$`(sobre C)`$\}$
- Ao analisar os estados que satisfazem o objetivo em G verifica-se que o conjunto de estados de objetivos é $S_g = \{s_2\}$

Todos os estados em S compartilham as proposições `(adjacente A B)`, `(adjacente B A)`, `(adjacente B C)` e `(adjacente C B)`. O plano que resolve o problem deste exemplo é $\rho = \{$`Mover(A B)`, `Mover(B C)`$\}$.

Apresentadas as definições básicas e essenciais sobre planejamento automático, falar-se-á nas seções 3.1.2 e 3.1.3 sobre os modelos

de planejamento utilizados na proposta deste trabalho para a obtenção de composições resilientes de Web services semânticos. Tal proposta será apresentada em detalhes no capítulo 4.

3.1.2 Planejamento Satisfazível

Partindo do princípio de que todo problema em NP pode ser reduzido ao problema da satisfazibilidade booleana (SAT) (BIERE et al., 2009), os pesquisadores Kautz e Selman (1992) propuseram um meio formal (ver definição 3.8) para mapear problemas de planejamento em problemas SAT. O mapeamento consiste, basicamente, em traduzir um problema de planejamento Σ em uma fórmula lógica proposicional ψ em uma forma normal conjuntiva (FNC) e testar a sua satisfazibilidade em relação a um *makespan* N (quantidade permitida de ações que o agente pode executar linearmente para alcançar os seus objetivos).

Definição 3.7. *(BIERE et al., 2009) Uma fórmula lógica proposicional ψ está em uma FNC se ela é uma conjunção de cláusulas, i.e. $\psi = C_1 \wedge C_2 \wedge \ldots \wedge C_z$, onde cada cláusula é uma disjunção de literais (símbolos proposicionais ou as suas negações), i.e, $C_j = (l_1 \vee l_2 \vee \ldots \vee l_m)$, com $1 \leq j \leq z$, $l_i = p$ ou $l_i = \neg p$ com $l_i \in L$, $p \in \mathbb{P}$, $L = \{l_1, l_2, \ldots, l_{2n}\}$, $|\mathbb{P}| = n$ e $m \leq 2n$.*

A transformação de uma fórmula lógica proposicional ψ para a sua forma normal conjuntiva é realizada de acordo com os seguintes passos, onde α, β e φ são literais (TINELLI, 2010):

1. Remover símbolos bicondicionais
   $(\alpha \leftrightarrow \beta) \leftrightarrow ((\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha))$

2. Remover símbolos de implicação
   $(\alpha \rightarrow \beta) \leftrightarrow (\neg\alpha \vee \beta))$

3. Aplicar as leis de Demorgan e Dupla Negação
   $\neg(\alpha \wedge \neg\beta) \leftrightarrow (\neg\alpha \vee \beta)$

4. Aplicar a distributividade da disjunção sobre a conjunção
   $\alpha \vee (\beta \wedge \varphi) \leftrightarrow (\alpha \vee \beta) \wedge (\alpha \vee \varphi)$

A maioria dos solucionadores SAT adota, como padrão de entrada, o formato DIMACS para representar FNCs, pois este formato permite representar as FNCs de um modo mais simples. Uma fórmula

lógica descrita no formato DIMACS possui a seguinte estrutura (MAKHORIN, 2011):

COMENTÁRIOS p FORMATO LIT CL FÓRMULA

Onde, COMENTÁRIOS denota um conjunto opcional de comentários sobre o problema devendo ser iniciado pela letra c, FORMATO refere-se ao formato da representação do problema (podendo ser na forma normal conjuntiva ou na forma normal disjuntiva[3]), LIT e CL denotam, respectivamente, o número de literais e cláusulas do problema. Se FÓRMULA refere-se a uma FNC, então os literais serão substituídos por inteiros positivos ou negativos, as disjunções serão representadas por espaços em branco, e as conjunções serão representadas por zeros.

Supondo que $\psi = (p_1 \vee \neg p_2) \wedge (\neg p_2 \vee \neg p_3) \wedge (\neg p_1 \vee \neg p_3) \wedge (\neg p_1 \vee \neg p_2)$ é uma fórmula lógica proposicional na sua FNC, então a sua representação no formato DIMACS, com $p_1 = 1$, $p_2 = 2$ e $p_3 = 3$, será:

```
c   um exemplo
p   cnf 3 4
1   -2   0
-2  -3   0
-1  -3   0
-1  -2   0
```

Ao receber ψ, o solucionador SAT buscará por um modelo σ, isto é, uma atribuição de valores verdade que torne a fórmula lógica semanticamente verdadeira (ou seja, uma atribuição de valores verdade que a satisfaz). Geralmente, tal atribuição é expressa na forma de uma lista constituída pelos inteiros que representam os literais da FNC. Nesta lista, os inteiros positivos representam os literais para os quais o valor semântico foi atribuído como verdadeiro, ao passo que os inteiros negativos representam os literais para os quais o valor semântico foi atribuído como falso. Para o exemplo anterior, uma possível atribuição de valores verdade é σ = {-1, -2, -3}, significando que, para σ ser satisfazível, os valores verdade de p_1, p_2 e p_3 devem ser todos falsos.

Caso um modelo σ seja encontrado, então é possível extrair de ψ um plano para o problema de planejamento. Se existirem k mode-

[3]Assim como a FNC, a forma normal disjuntiva (FND) representa um tipo de padrão para descrição de fórmulas lógicas proposicionais. Uma fórmula lógica está em uma FND se ela for uma disjunção de cláusulas e as cláusula forem conjunções de literais.

los para a fórmula lógica, então existem k planos, um plano para cada modelo σ encontrado. Se nenhum modelo for encontrado, então a fórmula lógica proposicional é dita insatisfazível em relação ao *makespan* N. Neste caso, o problema de planejamento pode ser reduzido a SAT para diferentes *makespans* com valores que são incrementados iterativamente à medida que fórmulas lógicas insatisfazíveis são obtidas. O processo de mapeamento se repete até que se encontre um *makespan* para o qual a fórmula lógica é satisfazível ou quando um número máximo de reduções seja alcançado (este caso garante que o processo de redução pare) (GIUNCHIGLIA; MARATEA, 2007).

Definição 3.8. *(KAUTZ; SELMAN, 1992) A tradução de um problema de planejamento Σ para SAT, considerando um* makespan *N, tal que $0 \leq t < N$, pode ser realizada pela seguinte função de mapeamento, tal que:*

$$f(\Sigma, N) = AEI_0 \wedge (\bigwedge_{0 \leq t < N} AES_t \wedge APC_t \wedge AEM_t) \wedge AO_N$$

Onde, AEI, AES, APC, AEM e AO, referem-se aos axiomas do estado inicial do mundo*,* estados sucessores*,* pré-condições*,* exclusão mútua *e* objetivos*, respectivamente.*

A seguir, os axiomas da definição 3.8 são descritos em detalhes. Considere que os axiomas AES, APC e AEM são constituídos por conjunções e/ou disjunções de literais na forma $p_{i,t}$ e $a_{j,t}$, onde $p_i \in \mathbb{P}$, $a_j \in A$ e $t \in [0, N)$ refere-se a um instante de tempo para determinar se o fato que o literal representa é verdadeiro naquele instante. Os axiomas AEI e AO são constituídos por literais da forma $p_{i,0}$ e $p_{i,N}$, respectivamente.

Definição 3.9. *(***Axioma de Estado Inicial - AEI***) O estado inicial do mundo $s_0 \in S$ é mapeado da seguinte forma:*

$$\bigwedge\{p_{i,0} \mid p_{i,0} \in s_0\} \wedge \bigwedge\{\neg p_{i,0} \mid p_{i,0} \in \mathbb{P} - s_0\}$$

A definição 3.9 mostra que os literais que fazem parte do estado inicial do mundo são indexados com o instante de tempo $t = 0$ que, por convenção, é o instante no qual inicia-se a busca por um plano.

Definição 3.10. *(***Axioma de Pré-condições - APC***) Para que uma ação $a_j \in A$ ocorra em um instante de tempo t é necessário que suas pré-condições estejam satisfeitas naquele mesmo instante, tal que:*

$$a_{j,t} \rightarrow \bigwedge\{p_{i,t} \mid p_{i,t} \in Pre(a_j)\}$$

Definição 3.11. *(**Axioma de Estados Sucessores - AES**) Um literal $p_{i,t} \in \mathbb{P}$ é verdadeiro no instante de tempo $t+1$ se e somente se:*

$$p_{i,t+1} \leftrightarrow ActCausesL \vee (p_{i,t} \wedge \neg ActCausesNotL)$$

onde, $ActCausesL$ refere-se à disjunção entre ações que tem, em seus conjuntos de adição (Add), o literal $p_{i,t}$ (ou seja, tornam p_i verdadeiro no instante de tempo $t+1$), e $ActCausesNotL$ refere-se à disjunção entre as ações que tem, em seus conjuntos de deleção (Del), o literal $p_{i,t}$.

A definição 3.11 mostra que um literal $p_{i,t} \in \mathbb{P}$ é verdadeiro no instante de tempo $t+1$ se, e somente se, alguma ação que o possui em seu conjunto *Add* ocorre no tempo t, ou se tal literal já é verdadeiro no tempo t e nenhuma ação que a possui em seu conjunto *Del* ocorre neste mesmo tempo.

Definição 3.12. *(**Axioma de exclusão mútua- AEM**) Para um dado instante de tempo t, no máximo uma ação pode ocorrer. Para cada par de ações a_j e a_k, com $j \neq k$, tem-se $a_j \rightarrow \neg a_k \leftrightarrow \neg a_j \vee \neg a_k$, tal que:*

$$\neg a_{j,t} \vee \neg a_{k,t}$$

A definição 3.12 mostra que a ocorrência de ações é um evento mutuamente exclusivo, logo o agente de planejamento pode executar apenas uma ação por vez.

Definição 3.13. *(**Axioma de Objetivos - AO**) O conjunto de objetivos do agente de planejamento $G \subseteq \mathbb{P}$ é mapeado da seguinte forma:*

$$\bigwedge\{p_{i,N} \mid p_{i,N} \in G\} \wedge \bigwedge\{\neg p_{i,N} \mid p_{i,N} \in \mathbb{P} - G\}$$

onde $p_{i,N}$ são os literais em $\mathbb{P}$ contidos em G assumindo-se que os objetivos são alcançados no tempo N.

Os efeitos de uma ação $a \in A$ executada no instante de tempo t findarão no instante de tempo $t+1$, assim, os efeitos da última ação de um plano que soluciona um problema de planejamento, que é executada no tempo $t = N$-1, findarão no tempo N, instante no qual os objetivos do agente de planejamento são mapeados na definição 3.13.

Através de um modelo σ, encontrado por um solucionador SAT, um plano ρ para o problema de planejamento pode ser obtido como uma sequência linear de ações $(a_{1,0}; a_{2,1}; \ldots; a_{N,N\text{-}1})$, onde o literal $a_{j,t}$ refere-se à ação $a_j \in A$ a ser executada no tempo t. Tal sequência é obtida recuperando-se as ações em A cujos literais da fórmula lógica estão representados por inteiros positivos no modelo σ e, em seguida, ordenando-as de forma crescente de acordo com os seus tempos de execução.

3.1.3 Planejamento Não-Determinístico

Planejamento não-determinístico (BERCHER; MATTMÜLLER, 2009) é uma extensão do planejamento clássico onde, em sua forma mais básica[4], as ações são constituídas por l pares de efeitos não-determinísticos, onde cada par (*Add*,*Del*) é constituído por uma lista de efeitos positivos (Add) e uma lista de efeitos negativos (Del). Após uma ação não-determinística ser executada, apenas um de seus pares de efeitos irá acontecer. Neste sentido, não existe certeza sobre qual estado do mundo será alcançado após o agente executar uma ação não-determinística, pois cada par de efeitos leva o agente a um diferente estado.

Definição 3.14. *O domínio de um problema de planejamento não-determinístico com observação total é uma 4-tupla* $\mathbb{D}_{ND} = \langle \mathbb{P}, S, A_{ND}, \gamma' \rangle$, *tal que:*

- $\mathbb{P} = \{p_1, p_2, \ldots, p_n\}$ *denota um conjunto de símbolos proposicionais, onde* $p \in \mathbb{P}$ *é um símbolo que identifica um fato do mundo (i.e, uma proposição).*
- $S = \{s_1, s_2, \ldots, s_m\}$ *denota um conjunto de estados, onde* $S \subseteq 2^{\mathbb{P}}$, *e* $s \in S$ *é um conjunto sobre* $\mathbb{P}$.
- $A_{ND} = \{a_1, a_2, \ldots, a_k\}$ *denota um conjunto de ações não-determinísticas, onde* $A_{ND}(s)$ *refere-se ao conjunto de ações não-determinísticas aplicáveis em* $s \in S$.
- $\gamma' : S \times A \to 2^S$ *denota uma função de transição não-determinística entre estados.*

Um problema de planejamento não-determinístico é observável quando o agente de planejamento é capaz de identificar todos os fatos

[4] Variantes de planejamento não-determinístico incluem estado inicial desconhecido, observabilidade parcial, ações com efeitos probabilísticos, dentre outros.

relevantes que compõem os estados do mundo para a concepção de um plano. Se o agente é capaz de observar apenas parte de tais fatos, diz-se que o problema de planejamento é parcialmente observável. A incapacidade do agente de observar todas as verdades que compõem os estados do mundo podem estar relacionadas a diferentes causas como, por exemplo, quantidade insuficiente de sensores ou sensores imprecisos para captar os fatos (ETZIONI et al., 1992).

Definição 3.15. *Uma instância de um problema de planejamento sobre um domínio não-determinístico $\mathbb{D}_{ND}$ é definido uma tupla $\Sigma_{ND} = \langle s_0, G \rangle$, tal que:*

$s_0 \in S$ *denota o estado inicial do mundo sobre o qual o agente inicia sua busca por um plano.*

$G \subseteq \mathbb{P}$ *denota um conjunto de objetivos do agente.*

Basicamente, as principais diferenças entre o planejamento clássico e o planejamento não-determinístico estão nas representações das funções de transição de estados determinística γ e não-determinística γ' (conforme as definições 3.1 e 3.14) e nos tipos de ações executáveis pelo agente, determinísticas e não-determinísticas (conforme as definições 3.4 e 3.16).

Definição 3.16. *Uma ação $a_{ND} \in A_{ND}$ com efeitos não-determinísticos e custo de execução é uma 5-tupla $a_{ND} = \langle Nome, Args, Pre, ND, \mathcal{K} \rangle$ tal que:*

$Nome$ *denota o nome da ação a_{ND}.*

$Args = \{x_1, x_2, \ldots, x_n\}$ *denota uma lista de argumentos, onde um parâmetro $x \in Args(a_{ND})$ refere-se a um objeto do mundo sobre o qual a ação irá operar.*

$Pre = \{p_1, p_2, \ldots, p_m\}$ *denota um conjunto de pré-condições que devem ser satisfeitas para que a ação a seja aplicável em um estado $s \in S$ do mundo.*

$ND = \{(Add, Del)_1, (Add, Del)_2, \ldots, (Add, Del)_l\}$ *refere-se a um conjunto formado por pares de efeitos não-determinísticos, onde um par $(Add, Del)_i$ representa, respectivamente, uma lista de efeitos positivos Add e uma lista de efeitos negativos Del, tal que, $Add, Del \subseteq \mathbb{P}$ e $Add \cap Del = \emptyset$.*

$\mathcal{K}$ *refere-se ao custo de execução da ação.*

Os componentes de uma ação $a_{ND} \in A_{ND}$ serão identificados, ao longo deste texto, por $Pre(a_{ND})$, $Add(a_{ND})$, $Del(a_{ND})$ e $\mathcal{K}(a_{ND})$. Após uma ação a_{ND} executar em um estado $s \in S$, um estado $s' \in S$ é alcançado de modo não-determinístico, tal que $s' = (s - Del) \cup Add$, com $(Add, Del)_i \in ND$.

A descrição de problemas de planejamento não-determinísticos pode ser feita através da linguagem padrão NuPDDL (BERTOLI, 2002). Esta linguagem é uma extensão da linguagem PDDL com suporte a diferentes formas de não-determinismo (ações com efeitos não-determinísticos, incerteza sobre o estado inicial, observabilidade parcial e nula, dentre outros). A gramática da seção 3.1.1 também foi estendida para dar suporte a ações não-determinísticas[5] com custo de execução da seguinte forma:

```
<domain>               ::=  (define (domain <name>)
                            ...
                            <functions-def>
                            ...
<functions-def>        ::=  (:functions <functions-list>)
<functions-list>       ::=  (<fluent>)<functions-list>*
<fluent>               ::=  <name>
                            ...
<action-nd-def>        ::=  (:action <name>
                       ::=  :parameters (<parameters-list>)
                       ::=  <action-nd-def-body>)
<action-nd-def-body>   ::=  :precondition <preconditions>
                       ::=  :effect <effects-nd>
<preconditions>        ::=  (and <predicates-list>)
<effects-nd>           ::=  (oneof (and <predicates-list>
                                   (not <predicate-list>)?)+))
```

A representação de ações não-determinísticas em NuPDDL se dá através do uso da palavra reservada `oneof` para cada par de efeitos não-determinísticos. `oneof` serve para indicar que, após uma ação executar, apenas um dos pares de efeitos irá acontecer. O custo de execução de uma ação não-determinística pode ser feito através da criação de fluentes, ou seja, termos cujos valores são números reais, diferentemente dos predicados cujos valores são booleanos. Fluentes são criados pela

[5] Por não fazerem parte do escopo deste trabalho, as demais formas de não-determinismo em NuPDDL não são suportadas na extensão da gramática da seção 3.1.1.

regra `<functions-def>` e os seus custos são atribuídos pelo token `assign`. Como as ações podem ter custo de execução diferentes, o token `assign` é descrito individualmente para cada ação como sendo um de seus efeitos.

Exemplo 2. *Considere que a ação* `mover` *do exemplo 1 foi redefinida para suportar efeitos não-determinísticos e custo de execução. Suponha que, ao partir de uma* `célula x` *em direção a uma* `célula y`*, o robô pode alcançar a* `célula y` *ou permanecer na* `célula x` *por não ter bateria suficiente para tal fim. Suponha, também, que o* `custo` *de locomoção entre duas células distintas é igual a 2.*

Deste modo, a ação `mover` fica assim definida:

```
:action mover
:parameters (?x - célula ?y - célula)
:precondition (and (sobre ?x))
:effect (oneof (and (sobre ?x) (assign (custo) 0)); parte de efeitos 1
               (and (sobre ?y) (not (sobre ?x))
                    (assign (custo) 2))); par de efeitos 2
```

Definição 3.17. *A solução para um problema de planejamento não-determinístico com observabilidade total é uma política* π *que mapeia estados do mundo a ações, isto é,* $\pi : S_\pi \rightarrow A_{ND}$*, onde* $S_\pi \subseteq S$ *e* $\forall s \in S_\pi : \exists a \in A_{ND}$*, tal que* $(s, a) \in \pi$*.*

Uma política π é um plano do tipo condicional, isto é, um plano que indica qual ação deve ser executada a partir dos estados alcançados não-deterministicamente. A política π pode ser estruturada na forma de uma árvore, onde os vértices representam estados do mundo e as arestas representam transições não-determinísticas ocasionadas pela execução das ações. Ademais, uma política pode ser classificada como uma solução *fraca*, *forte* ou *forte cíclica* (CIMATTI et al., 2003). Na solução fraca não garante-se que os objetivos do agente serão alcançados quando for executada, já que para alguns estados em S_π pode não haver caminhos para algum estado de objetivos.

Na solução forte garante-se que os objetivos do agente sempre serão alcançados quando executada visto que para todo estado em S_π existe algum caminho para um estado de objetivos. Na solução forte cíclica garante-se que sempre haverá um caminho para um estado de objetivos para todo estado em S_π. Entretanto, ao executar a política, pode-se entrar em loop sobre uma sequência de estados enquanto efeitos desejados não são produzidos. Se for possível, ao produzir os efeitos

desejados, a política sai do loop e avança em direção ao estado de objetivos.

Neste trabalho, adota-se uma política forte cíclica como uma estratégia de contingência para a obtenção de composições resilientes de Web services. Como será visto no capítulo 4, tal política é estruturada na forma de uma árvore de decisão binária, na qual k planos (composições alternativas) são sobrepostos. Além do mais, a política é concebida com um número máximo de execuções de ciclos que partem e retornam ao estado inicial do mundo.

3.2 WEB SERVICES SEMÂNTICOS

Com o advento da Internet, o acesso à informação se tornou possível a qualquer momento e lugar. A Internet tem proporcionado uma série de benefícios aos seus usuários permitindo-os, por exemplo, interagir entre si em tempo real, realizar comércio eletrônico, estudar a distância, entre outros recursos. Entretanto, a rede mundial de computadores é formada por um conjunto de máquinas que possuem diferentes arquiteturas e sistemas operacionais, e para que as aplicações possam se comunicar neste ambiente heterogêneo é necessária a adoção de formatos e protocolos padronizados para que elas troquem mensagens (COSTA, 2007).

Web services permitem componentes de software distribuídos na Web se acoplarem dinamicamente e trocarem mensagens. Um Web service é um conjunto de funcionalidades disponíveis via Web por meio de uma API constituída de operações que dão acesso a tais funcionalidades. Cada operação tem parâmetros de entrada, parâmetros de saída, pré-condições e efeitos. A descrição, publicação, localização e invocação de operações, assim como a troca de mensagens entre Web services são especificadas usando padrões da Web definidos pelo World Wide Web Consortium[6] (W3C, 2004). Dentre estes padrões se encontram:

- A linguagem WSDL (*Web services Description Language*) que, basicamente, descreve de modo sintático as operações dos Web services.

- A descoberta e a publicação é feita pelo serviço de diretório UDDI (*Universal Description, Discovery and Integration*).

[6] Além dos Web services definidos pelos padrões da W3C, existem os Web services REST, que também são bastante utilizados no mercado.

- A estrutura padrão para as mensagens que serão trocadas é definida pelo protocolo SOAP (*Simple Object Access Protocol*).

- As mensagens SOAP são transportadas pelo protocolo HTTP (*Hyper Text Transport Protocol*).

Entretanto, a descrição puramente sintática das operações dá margem a más interpretações das funcionalidades dos Web services, já que operações com descrições sintáticas semelhantes podem possuir significados totalmente distintos, e operações com descrições sintáticas distintas podem possuir o mesmo significado (logo desempenham as mesmas funções). Além do mais, este tipo de descrição não fornece informações suficientes para que sempre seja possível que agentes inteligentes descubram automaticamente Web services para cumprir um conjunto de tarefas (BARTALOS; BIELIKOVA, 2011).

A adição de anotações semânticas nas descrições das operações (i.e, informações adicionais que definem o significado dos elementos que compõem as operações mapeando-os a conceitos ou instâncias formalmente descritas em ontologias de domínio) surge como uma alternativa para superar as limitações de expressividade das descrições sintáticas que podem ser especificadas pelos padrões de Web services convencionais (WSDL, SOAP, UDDI, etc.) ou Web services REST. Tais descrições semanticamente ricas de Web services podem ser especificadas usando linguagens com WSDL-S (*Web services Description Language with Semantics*), OWL-S (*OWL-based Service Ontology*) e WSMO (*Web Services Modeling Ontology*), e servem para suportar busca, seleção e composição automatizadas de Web services.

Ademais, os Web services estão vulneráveis a problemas de execução, isto é, podem executar de modo inapropriado, retornar resultados errados e insatisfatórios. Isto ocorre, principalmente, devido a problemas de natureza dinâmica e não-determinística das redes de computadores (i.e, problemas na infraestrutura de transmissão que podem ocasionar falhas de comunicação, servidores indisponíveis, etc), não cumprimento das pré-condições para que as operações sejam executadas, falhas de software e hardware, alterações nas interfaces dos serviços, passagem de parâmetros em número e tipos incorretos, descumprimento de parâmetros de qualidade, violações de SLA, dentre outros. Portanto, os objetivos do usuário podem não ser satisfeitos.

Sob estas circunstâncias, é importante levar em consideração que o comportamento dos Web services é intrinsecamente não-determinístico, muitas vezes inesperado e inadequado e, portanto, estão sujeitos a falhar em cumprir as funções para as quais foram designados

e não satisfazerem os objetivos do usuário (GIACOMO; PATRIZI, 2009; TRAVERSO; PISTORE, 2004). Mais especificamente, uma falha (também chamada de defeito) pode ser vista como o não atendimento de um conjunto de funções. A causa de uma falha é chamada de *falta* (como os problemas relacionados às redes de computadores anteriormente descritos, por exemplo) . Uma falta produz erros que podem descarrilhar o fluxo correto (fluxo esperado) de execução de um processo, levá-lo a um estado de erro e impedi-lo de satisfazer com sucesso as suas funções. Caso isso ocorra, ações corretivas devem ser executadas para recuperar o processo da falha, ou seja, fazer com que o processo volte ao fluxo correto de execução (CHAN et al., 2009) .

A definição 3.18 baseia-se em (MARKOU; REFANIDIS, 2016) e reflete os conceitos da tecnologia de Web services usados nesta dissertação.

Definição 3.18. *Uma operação op de um Web service semântico w com comportamento não-determinístico é uma 5-tupla* $op = \langle I, O, Pre', ND', \mathcal{K}' \rangle$*, tal que:*

$I = \{i_1, i_2, \ldots, i_n\}$ *refere-se a um conjunto de parâmetros de entrada.*

$O = \{o_1, o_2, \ldots, o_m\}$ *refere-se a um conjunto de parâmetros de saída.*

$Pre' = \{p_1, p_2, \ldots, p_k\}$ *refere-se a um conjunto de pré-condições que devem ser satisfeitas para que exista a chance da operação produzir os efeitos desejados. Caso as pré-condições sejam satisfeitas e nenhum erro ocorra em tempo de execução, então a operação produzirá um conjunto de efeitos desejados. Caso algum problema de execução ocorra, então algum conjunto de efeitos indesejados será produzido.*

$ND' = \{\mathrm{Eff}_d, \mathrm{Eff}_u\}$ *refere-se a um conjunto de efeitos não-determinísticos, onde* Eff_d *denota um conjunto de efeitos desejados, isto é, efeitos que ocorrem quando a operação finaliza a sua execução com sucesso, e* Eff_u *denota um conjunto de efeitos indesejados, isto é, efeitos que ocorrem quando a operação não produz os efeitos desejados.*

$\mathcal{K}'$ *refere-se ao custo de execução da operação.*

Os componentes de uma dada operação *op* são identificados, ao longo deste texto, por $I(op)$, $O(op)$, $Pre'(op)$, $ND'(op)$ e $\mathcal{K}'(op)$. Diferentes formas de mensurar o custo de execução $\mathcal{K}'$ de operações vêm

sendo propostas. Dentre tais formas, o custo pode ser obtido diretamente das descrições do serviço (BENABOUD; MAAMRI; SAHNOUN, 2016; PAN; MAO, 2009), através de predição (CHEN et al., 2016b; WU et al., 2015) ou monitoramento e análise das execuções do serviços (ANGARITA; RUKOZ; MANOUVRIER, 2015; HASAN; JAAFAR; HASSAN, 2014).

Fórmulas lógicas são utilizadas para descrever as condições necessárias para que as operações tenham chance de executar corretamente e as condições sobre as quais os efeitos desejados e indesejados ocorrem. Basicamente, tais fórmulas associam os parâmetros de entrada e saída das operações aos valores esperados, e podem ser usadas por um motor de execução auxiliando-os a identificar quando uma operação falhou em obter os efeitos desejados.

Como muitos parâmetros são de tipos complexos (i.e, elementos em estruturas de dados), as fórmulas lógicas podem ser descritas utilizando os atributos de tais estruturas ou elementos, onde *parâmetroX.atributoY*, por exemplo, indica o acesso ao atributo Y do parâmetro X. Qualquer conectivo lógico (i.e, $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$), operadores de comparação e aritméticos podem ser utilizados para expressar condições.

Como não é possível prever todos os efeitos indesejados em Web services, uma forma de expressar o conjunto de efeitos indesejados Eff_u é denotando-o como o complemento do conjunto de efeitos desejados Eff_d. Neste sentido, qualquer efeito produzido por uma operação que não pertence a Eff_d certamente pertence a Eff_u. A fórmula lógica para Eff_u é construída a partir da fórmula lógica para Eff_d, onde substitui-se os operadores aritméticos e lógicos da fórmula de Eff_d por seus operados inversos (i.e, a soma é substituída pela subtração, o ou lógico é substituído pelo e lógico, por exemplo) obtendo, assim, uma fórmula para Eff_u.

Logo, para saber a qual conjunto pertence o efeito produzido após uma operação executar, basta que os valores associados aos parâmetros de saída contidos na mensagem de resposta SOAP sejam comparados aos valores explicitados nas fórmulas lógicas descritas para os efeitos desejados (KHOLY; FATATRY, 2016). Entretanto, em certas situações, pode-se não obter uma resposta por parte do Web service invocado (como ocorre quando o servidor está indisponível ou processos estão em estado de *deadlock*, por exemplo), logo não há produção de efeitos. Um mecanismo de *timeout* pode ser configurado para determinar o tempo máximo de espera por uma resposta. A ação a ser executada quando um conjunto de efeito indesejados for gerado, a fim de contornar problemas de execução, pode ser tomada quando o limite

de tempo de espera por uma resposta é extrapolado, pois esta também é uma situação na qual os efeitos desejados não foram produzidos.

Exemplo 3. *Suponha que uma certa operação libera acesso a streaming de vídeos de acordo com a localização geográfica do usuário e que um de seus parâmetros é o endereço do usuário que é um tipo complexo. Uma mensagem SOAP enviada ao Web service detentor da operação pode ser descrita de acordo com o seguinte XML Schema:*

```
<xs:complexType name="endereço">
   <xs:sequence>
     <xs:element name="país" type:"xs:string"/>
     <xs:element name="estado" type:"xs:string"/>
     <xs:element name="cidade" type:"xs:string"/>
     <xs:element name="CEP" type:"xs:int"/>
     <xs:element name="número" type:"xs:int"/>
   </xs:sequence>
</xs:complexType>
```

Uma fórmula lógica que descreve as pré-condições para que a operação seja executada pode ser:

$$endereço.país = \text{"Brasil"} \wedge endereço.estado = \text{"SC"}$$

Uma fórmula lógica para descrever os efeitos desejados pode ser:

$$autorização = \text{"concedida"} \wedge transmissão.estado = \text{"carregando"}$$

A fórmula lógica que descreve os efeitos indesejados é:

$$autorização \neq \text{"concedida"} \vee transmissão.estado \neq \text{"carregando"}$$

Ademais, os efeitos produzidos por uma operação podem estar relacionados a alterações no mundo real, como é o caso de uma transação de créditos que, quando bem sucedida, diminui o saldo do usuário (entre outros efeitos), ou à mera produção de informações, como ocorre quando uma consulta a uma base de dados de uma loja determina se a loja possui em estoque um determinado produto (neste caso a operação apenas produz saídas que indicam tal informação e não ocorrem alterações no mundo).

Alguns trabalhos como (HOFFMANN; WEBER; KRAFT, 2012; MARRELLA; MECELLA; SARDINA, 2014) diferenciam possíveis efeitos desejados e indesejados em n conjuntos, e dependendo do conjunto de efeitos indesejados obtido após a execução da operação, uma regra para recuperação de falha é aplicada baseando-se nos problemas de execução

e erros por eles gerados que impediram a execução da operação com sucesso. Entretanto, como visto no capítulo de introdução e como será abordado no capítulo 4, este trabalho tem como principal objetivo a obtenção de composições resilientes de Web services semânticos e, para tal, este trabalho se concentra na obtenção de várias composições alternativas e na sobreposição das mesmas em uma estrutura (árvore binária de decisão) que permita a um motor de execução trocar, rapidamente e com custo mínimo, uma composição por outra tão compatível quanto possível ao que já foi executado com sucesso, quando uma operação não gerar os efeitos desejados. Deste modo, aumentam-se as chances de que os objetivos do usuário sejam satisfeitos e evita-se o custo de auferir composições alternativas em tempo de execução.

Exemplo 4. *Considere que* `pagamentoViaCrédito` *é uma operação semântica de um certo Web service que permite realizar pagamento via crédito. Os elementos que compõem tal operação podem ser definidos da seguinte forma:*

$I = \{$`cartãoDeCrédito, valor`$\}$;
$O = \{$`transação`$\}$;
$Pre = \{$`cartãoDeCrédito.saldo` $\geq$ `valor`$\}$;
$ND = \langle\{$`transação.estado` $=$ `aprovado` $\wedge$
`cartãoDeCrédito.saldo` $=$ `cartãoDeCrédito.saldo-valor`$\}$,
$\{$`transação.estado` $\neq$ `aprovado` $\vee$
`cartãoDeCrédito.saldo` $\neq$ `cartãoDeCrédito.saldo-valor`$\}\rangle$;
$\mathcal{K} = 1$

Para o exemplo 4 foi definido como pré-requisito que deve haver saldo suficiente para que o pagamento seja efetuado. O conjunto de efeitos desejados indica que a transação precisa ser aprovada e o saldo do cartão atualizado. O conjunto de efeitos indesejados indica qualquer situação em que um dos efeitos desejados não é alcançado.

Se em uma dada ontologia há uma conceitualização (descrições formais de conceitos, seus atributos e suas relações) para *cartão de crédito*, *valor* e *transação* (como, por exemplo, as classes *CartãoDeCrédito*, *Preço* e *Transação*, respectivamente, ilustradas na ontologia da figura 3), então, pode-se usar tais conceitos da ontologia para definir os tipos dos parâmetros da operação `pagamentoViaCrédito`.

Na figura 3 os atributos das classes foram omitidos por simplicidade e facilidade de compreensão. O símbolo $\equiv$ é utilizado para indicar que duas classes (conceitos) são equivalentes, onde as relações pertencentes à classe do lado direito da relação foram ocultadas.

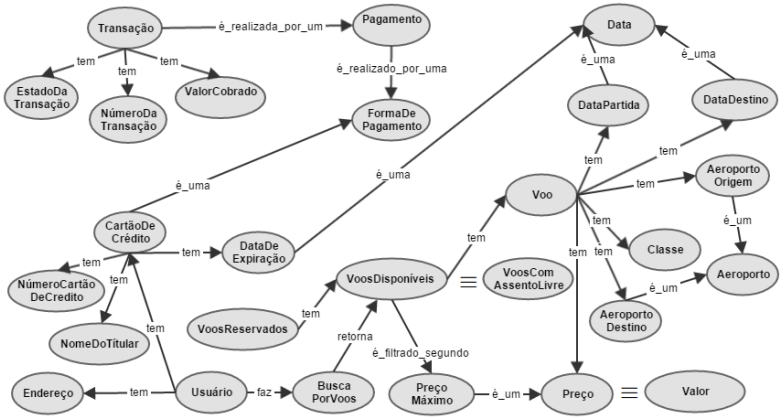

Figura 3 – Trecho de ontologia sobre passagens aéreas.

Linguagens como OWL-S, WSDL-S, WSMO, dentre outras, podem ser utilizadas para descrever semanticamente os Web services, seus componentes (e.g., operações, parâmetros de entrada e saída, pré-condições) e os seus comportamentos (e.g., efeitos). Basicamente, estas linguagens associam os componentes dos Web services à classes e objetos (instâncias de classes) de ontologias e bases de conhecimento. Neste sentido, as ontologias permitem que raciocinadores semânticos façam associações e inferências sobre os componentes dos Web services automaticamente. Deste modo, pode-se determinar quando parâmetros sintaticamente distintos são semanticamente equivalentes e, assim, determinar, por exemplo, quando uma operação produz saídas que podem ser consumidas como entradas por outras operações.

3.2.1 Composições de Web Services Semânticos

A composição de Web services semânticos tem como objetivo combinar um conjunto de operações de tais serviços para formar serviços mais complexos e sofisticados a fim de atender requisições de usuário para as quais não existem Web services que possam satisfazê-las. Basicamente, para que uma composição de Web services seja obtida é necessário *descobrir* um conjunto de Web services e operações que atendam parcialmente os objetivos do usuário, *selecionar* as operações mais apropriadas segundo um conjunto de critérios, determinar como

as invocações das operações estarão organizadas (i.e, um workflow), *executar* a composição resultante e monitorar tal execução (LEMOS; DANIEL; BENATALLAH, 2015).

Definição 3.19. *(ZOU et al., 2014) Uma requisição de usuário é uma requisição na qual o usuário solicita um serviço e pode ser definida como uma tupla $r = \langle I, O \rangle$, tal que:*

$I = \{i_1, i_2, \ldots, i_n\}$ *refere-se a um conjunto de parâmetros de entrada fornecidos pelo usuário necessários para que uma ou mais operações de Web services possam ser executadas.*

$O = \{o_1, o_2, \ldots, o_m\}$ *refere-se a um conjunto de parâmetros de saída que o usuário deseja obter.*

Os componentes de uma requisição r serão identificados, ao longo deste texto, por $I(r)$ e $O(r)$.

Exemplo 5. *Suponha que um usuário deseja um Web service para realizar pagamento via crédito, então uma requisição por uma operação que realize tal tarefa pode ser definida da seguinte forma:*

$I(r) = \{$`cartãoDeCrédito, valor`$\}$

$O(r) = \{$`transação`$\}$

Uma operação encontrada para tal fim poderia ser a operação `pagamentoViaCrédito` apresentada no exemplo 4.

Definição 3.20. *(BARTALOS; BIELIKOVA, 2011) Seja $W = \{w_1, , w_2, \ldots, w_n\}$ um repositório de Web services semânticos. Uma composição de Web services semânticos é um tupla $\mathcal{C} = (W', R)$, tal que:*

$W' = \{w_1, w_2, \ldots, w_n\}$*, com $W' \subseteq W$, denota um conjunto de Web services com operações que são compatíveis, diretamente ou indiretamente, com a requisição do usuário.*

$R \subseteq O_{op_i} \times I_{op_j}$ *denota as relações entre parâmetros de saída e parâmetros de entrada das operações dos Web services em W', onde O_{op_i} refere-se aos parâmetros de saída produzidos pela operação op_i e I_{op_j} refere-se aos parâmetros de entrada da operação op_j. Neste sentido, uma relação (O_{op_i}, I_{op_j}) indica que os parâmetros de saída produzidos pela operação op_i são consumidos como entrada pela operação op_j, com $op_i \neq op_j$. Alguns parâmetro não conectados servem como entradas e saídas da composição.*

Motores de busca como *SESA* (FENSEL; KERRIGAN; ZAREMBA, 2008) e *SADI* (WILKINSON; VANDERVALK; MCCARTHY, 2011) podem ser usados para obter o conjunto de Web services $W' \subseteq W$. Basicamente, estes motores analisam quais operações possuem parâmetros que combinam (sintática ou semanticamente) com os parâmetros fornecidos na requisição de usuário. Diferentes e corretas composições das operações em W', quando possíveis, formam diferentes composições de Web services.

Exemplo 6. *Considere que um usuário deseja comprar passagens aéreas para viajar entre duas cidades e que não exista Web service em W capaz de realizar tal tarefa. Suponha, também, que a requisição do usuário é composta pelo conjunto de parâmetros de entrada I_r =* {`dataPartida, dataRetorno, aeroportoOrigem, aeroportoDestino, classe, cartãoDeCrédito, preçoMáximo`} *onde* `preçoMáximo` *refere-se ao valor máximo que o usuário pretende pagar nas passagens aéreas, e o parâmetro de saída desejado O_r=*{`transação`}* representando o que o usuário espera como efeito da aquisição das passagens. Suponha que as operações em $W' \subseteq W$ que satisfazem parcialmente a requisição do usuário são obtidas por um motor de descoberta de Web services semânticos, que os tipos dos parâmetros são referências às classes apresentadas na figura 3 (onde, neste caso, por simplicidade, mas sem perda de generalidade os nomes dos parâmetros coincidem com os nomes das classes), e que o raciocinador semântico infere que os parâmetros* `voosDisponíveis` *e* `voosComAssentoLivres`*, e os parâmetros* `preço` *e* `valor` *são semanticamente equivalentes, logo as operações a serem usadas são as seguintes:*

`ListarVoosDisponíveis (LVDop)`: *retorna uma lista de voos disponíveis de acordo com os parâmetros fornecidos.*
$I =$ {`dataPartida,dataRetono,aeroportoOrigem,aeroportoDestino,classe`};
$O =$ {`voosDisponíveis`};
$Pre = \{\}$;
$ND = \langle$ {`voosDisponíveis.tamanho`>0},{`voosDisponíveis.tamanho`$=0$}$\rangle$;
$\mathcal{K} = 1$.

`ReeservarVoosDisponíveis (RVDop)`: *reserva dois voos onde a soma dos custos das passagens é de até* `preçoMáximo`.
$I =$ {`voosComAssentoLivres,preçoMáximo`};
$O =$ {`voosReservados,preço`};
$Pre =$ {`voosComAssentoLivres.tamanho`> 0};
$ND = \langle$ {`voosReservados.tamanho`> 0}, {`voosDisponíveis.tamanho`$=0$}$\rangle$;
$\mathcal{K} = 1$.

`PagamentoViaCrédito (PVCop)`: *realiza pagamento via crédito:*
$I = \{$`cartaoDeCrédito, valor`$\}$;
$O = \{$`transação`$\}$;
$Pre = \{$`cartaoDeCrédito.saldo` $\geq$ `valor`$\}$;
$ND = \langle\{$`transação.estado` $=$ `aprovado` $\wedge$
`cartaoDeCrédito.saldo` $=$ `cartaoDeCrédito.saldo`$-$`valor`$\}$,
$\{$`transação.estado` $\neq$ `aprovado` $\vee$
`cartaoDeCrédito.saldo` $\neq$ `cartaoDeCrédito.saldo`$-$`valor`$\}\rangle$;
$\mathcal{K} = 2$.

Uma possível composição de Web services poderia ser formada pela seguinte ordenação de operações:

1. Inicialmente, invoca-se a operação `LVDop` passando como argumento os parâmetros `dataPartida,dataRetono,aeroportoOrigem, aeroportoDestino,classe`;
2. Em seguida, invoca-se a operação `RVop` que obtem de `LVDop` o parâmetro `voosDisponíves` e `preçoMáximo` da requisição.
3. E, finalmente, invoca-se a operação `PVCop` obtendo os parâmetros `cartãoDeCrédito` da requisição fornecidos pelo usuário e o parâmetro de saída `valor` produzido por `RVop`.

Conforme definição 3.20, o conjunto de ligações entre as operações é definido como $R = \{$(`LVDop`,`RVDop`),(`RVDop`,`PVCop`)$\}$. A figura 4 ilustra um grafo que representa o fluxo de controle da composição do exemplo 6.

Figura 4 – Grafo de fluxo de controle.

Como visto na seção anterior, Web services estão sujeitos a não cumprirem as funções para as quais foram designados devido à ocorrência não-determinística de problemas de execução. Portanto, os objetivos do usuário podem não ser satisfeitos por uma composição fixa sem plano de contingência. Neste sentido, é fundamental que a composição obtida seja resiliente para que, ao ser executada em um ambiente real (i.e, dinâmico, não-determinístico e muitas vezes instável), possa aumentar as chances de que a sua execução finde com sucesso.

Resiliência é a capacidade de um sistema de contornar problemas de execução e fornecer seus serviços de modo contínuo frente a ocorrência de falhas em ambientes dinâmicos e não-determinísticos. Neste sentido, ao alcançar um estado de erro no qual satisfazer os objetivos se tornar improvável, o sistema deve ser capaz de voltar a um estado consistente dando continuidade ao fluxo esperado de execução (CABRI, 2014; LAPRIE, 2008).

Uma forma de se alcançar resiliência em composições de Web services é através da obtenção de composições alternativas que poderão ser obtidas, por exemplo, variando-se as operações que realizam de modo similar uma mesma tarefa. A obtenção de operações similares é factível, pois o número de Web services disponíveis na Web é grande e tem crescido cada vez mais nos últimos anos, logo existe uma alta quantidade de serviços que realizam, de modos similares, as mesmas funções (CHEN et al., 2016a).

Definição 3.21. *(CARDOSO, 2007) Duas operações de Web services, op_i e op_j, são similares se ambas operações são funcionalmente equivalentes e seus componentes são estruturalmente e semanticamente compatíveis. Neste sentido, a operação op_i pode ser executada alternativamente quando a operação op_j não produzir os efeitos desejados, se:*

$$\begin{array}{ccccc} O_{op_i} \subseteq O_{op_j} & \wedge & I_{op_j} \subseteq I_{op_i} & \wedge \\ ND_{op_i} \subseteq ND_{op_j} & \wedge & Pre_{op_j} \subseteq Pre_{op_i} & \end{array}$$

Devido à alta disponibilidade de operação similares, diferentes métodos têm sido propostos para se obter de modo eficiente o melhor Web service com a operação desejada. O melhor Web service é determinado através da análise de um conjunto de critérios como, por exemplo, custo e tempo de execução, disponibilidade, confiabilidade, reputação, dentre outros. Exemplos de métodos para determinar o melhor Web service podem ser encontrados em (CHEN et al., 2016b; HELALI; AZZOUNA; GHEDIRA, 2016; WANG; ZHENG, 2016; MA et al., 2016; WU et al., 2015).

No próximo capítulo é apresentada a proposta deste trabalho para se obter composições resilientes de Web services.

4 PROPOSTA

Este capítulo descreve a abordagem que o presente trabalho adota para a obtenção de composições resilientes de Web services. Esta abordagem combina planejamento não-determinístico e SAT (satisfazibilidade booleana) para que k planos determinísticos alternativos (composições alternativas de Web services) que solucionam uma dada requisição sejam obtidos. Posteriormente, os planos obtidos são fundidos em uma estratégia de contingência na forma de uma árvore de decisão binária (composição resiliente resultante da fusão das composições alternativas) que permite aos motores de execução de processos lidar com falhas mediante a rápida seleção, com custo mínimo, de uma composição alternativa e compatível com aquela que falhou. Deste modo, aumentam-se as chances de que os objetivos do usuário sejam satisfeitos.

A figura 5 ilustra um framework que contempla os módulos e as fases de processamento para gerar que a referente estratégia de contingência proposta neste trabalho seja obtida. Em síntese, as fases são as seguintes: (i) na fase de *pré-processamento*, mapeia-se o problema de obter uma composição de Web services para uma dada requisição de usuário a um problema de planejamento não-determinístico que, por sua vez, é mapeado a um problema de planejamento determinístico; (ii) na fase de *planejamento*, encontra-se k planos determinísticos que solucionam o problema de planejamento obtido na fase (i) e os combina em uma árvore de decisão binária; (iii) na fase de *execução*, transcreve-se a árvore de decisão binária em um processo executável para que a estratégia de contingência possa ser executada por um motor.

4.1 FASE DE PRÉ-PROCESSAMENTO

4.1.1 Módulo Motor de Descoberta

A fase de pré-processamento inicia-se pelo módulo *Motor de Descoberta*, que *seleciona* um conjunto de Web services semânticos $W' \subseteq W$[1] que atendam totalmente ou parcialmente uma requisição de usuá-

[1] W, como visto na definição 3.20, refere-se a um conjunto de Web services semânticos em um repositório. Qualquer linguagem para descrição semântica de serviços, como, por exemplo, OWL-S, WSMO, WSDL-S, dentre outras, pode ser usada para descrever os Web services em W. Ademais, diversos métodos da literatura podem

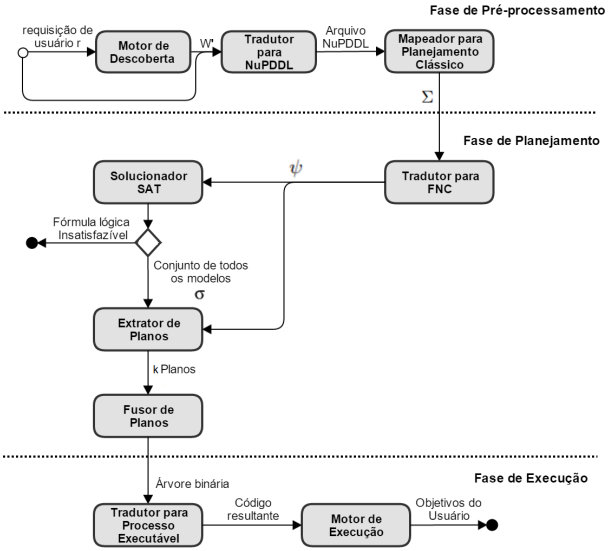

Figura 5 – Fases e módulos de processamento do framework proposto.

rio. Então, para cada conjunto de parâmetros semanticamente equivalentes, *reescreve* os nomes dos parâmetros, dos seus respectivos tipos e dos elementos que constituem as fórmulas condicionais (onde as fórmulas condicionais têm como base o formato $parâmetroX.atributoY$) que são equivalentes para um mesmo nome de parâmetro, um mesmo nome de tipo e um mesmo nome de elemento, respectivamente. Por exemplo, considerando que os parâmetros `voosDisponíveis` e `voosComAssentoLivres` das operações `ListarVoosDisponíveis` e `ReeservarVoosDisponíveis`, respectivamente, apresentadas no exemplo 6 sejam semanticamente equivalentes. Então, ambos os parâmetros passarão a ser identificados por um único nome assim como os seus tipos. Os elementos que constituem as fórmulas condicionais de tais operações também são reescritos para acomodar elementos com os mesmos nomes. Neste sentido, os parâmetros anteriormente citados poderão ser reescritos para `voosDisponíveis`, seus tipos reescritos para `VoosDisponíveis`, e, por fim, a fórmula condicional `voosComAssentoLivres.tamanho>0` é reescrita para `voosDisponíveis.tamanho>0`.

Deste modo, na fase de planejamento, um único símbolo proposicional p será criado para parâmetros semanticamente equivalentes e

ser usados para selecionar Web services de modo a obter W'.

um único símbolo proposicional p' será criado para as fórmulas condicionais semanticamente equivalentes. Entretanto, sem a reescrita das operações em W' e de suas respectivas fórmulas condicionais, o conjunto $\mathbb{P}$ resultante será maior do que o necessário, pois conterá distintos símbolos proposicionais para representar uma mesma informação. Neste sentido, por exemplo, se houverem n parâmetros sintaticamente distintos porém semanticamente equivalentes (o que significa que tais parâmetros representam a mesma informação), então serão criados n símbolos proposicionais para representá-los na fórmula lógica proposicional ψ resultante da tradução de planejamento para SAT.

Além do mais, sem a reescrita das operações, o agente de planejamento nem sempre estará apto a identificar quando uma ação $a \in A$ poderá ser executada em um determinado estado $s \in S$, pois, mesmo que os fatos necessários para que a ação a ocorra façam parte do estado s, os símbolos que os representam podem ser sintaticamente distintos daquelas em $Pre(a)$, logo o agente de planejamento entenderá que as pré-condições para que a ação a seja executada não estão satisfeitas no estado s.

Como resultado, o mapeamento de planejamento para SAT derivará mais axiomas do que o processo com reescrita, onde vários destes axiomas possuirão o mesmo significado (informações redundantes), implicando em perda de desempenho por parte do solucionador SAT que terá mais trabalho em determinar a satisfazibilidade da fórmula lógica proposicional ψ resultante. Além do mais, mesmo que o problema tenha solução, o solucionador SAT, assim como o agente de planejamento, poderá não encontrar nenhum modelo que satisfaça a fórmula lógica ψ a ser obtida.

Com a reescrita, sempre haverá garantia de que um certo fato será representado por um único símbolo proposicional. Assim, os n parâmetros passarão a ser identificados por apenas um único nome e durante a tradução de planejamento para SAT um único símbolo proposicional será criado para representá-los. Deste modo, o agente de planejamento se torna apto a identificar, com perfeição, quando poderá executar uma ação em um estado, pois o conjunto de símbolos que representa as suas pré-condições será único, e se tais pré-condições estiverem satisfeitas, então o conjunto de símbolos estará presente naquele estado.

4.1.2 Módulo Tradutor para NuPDDL

Na segunda parte da fase de pré-processamento, o conjunto W' e a requisição r do usuário são enviados ao módulo *Tradutor para NuPDDL* que, por sua vez, obtém em NuPDDL, conforme a gramática estendida apresentada na seção 3.1.3, uma versão em planejamento não-determinístico do problema de composição de Web services em questão. O problema resultante em NuPDDL é descrito em um arquivo de saída. Para isso, a seguinte convenção, inspirada em (WANG et al., 2015; ZOU et al., 2014), é adotada para mapear os elementos envolvidos no problema de composição de Web services em um problema de planejamento não-determinístico:

Descrição do domínio:

- Devido à versão do NuPDDL adotada neste trabalho ser fortemente tipada, os parâmetros, pré-condições e efeitos das operações em W' deverão ser mapeados aos seus respectivos elementos em NuPDDL considerando que possuem tipos. Para uma certa operação *op*, os tipos de seus parâmetros serão aqueles descritos no seu perfil (classes de ontologias). As pré-condições, efeitos desejados e efeitos indesejados serão considerados como sendo dos tipos `Condição_x`, onde `x` $\in N$. O valor de `x` se diferenciará para cada um destes elementos, neste sentido cada elemento terá o seu próprio tipo criado automaticamente pelo compilador na fase de pré-processamento. Por sua vez, cada tipo `Condição_x` será considerado subtipo de `Fato`. Tais tipos irão aparecer no escopo de `Type` do arquivo NuPDDL resultante.

- Predicados serão criados com aridade um e identificados por `fato(?x - Fato)`, onde `fato` refere-se ao nome do predicado e `?x` (um fato que irá constituir um estado do mundo) pode indicar que um parâmetro de entrada ou saída ($I(op)$ e $O(op)$) de uma certa operação *op* está disponível ou que um efeito desejado ou indesejado (Eff_d e Eff_u) foi produzido por *op* após a sua execução. Tais predicados irão aparecer no escopo de `predicates` do arquivo NuPDDL resultante.

- Para cada operação *op* de um Web service $w \in W'$ uma ação não-determinísticas $a_{ND} \in A_{ND}$ é obtida (referida por `action` no escopo de `domain` do arquivo NuPDDL resultante), neste sentido:

  – O nome da operação se torna o nome da ação a_{ND}.

- O custo de execução $\mathcal{K}'(op)$ se torna o custo de execução $\mathcal{K}(a_{ND})$. Este custo é representado pelo fluente `custo` que, por sua vez, é um dos efeitos de execução de a_{ND}.
- Os parâmetros de entrada e saída, pré-requisitos e efeitos se tornam os parâmetros de entrada da ação a_{ND}, os quais serão considerados os objetos manipuláveis pelo agente de planejamento.
- Os pré-requisitos de execução da ação a_{ND} se tornam os pré-requisitos da operação *op* unidos com os seus parâmetros de entrada, tal que, $Pre(a_{ND}) = I(op) \cup Pre'(op)$. Assim, para que tal ação ocorra em um estado $s \in S$ é necessário que os parâmetros de entrada da operação que a originou estejam disponíveis naquele estado.
- A ação a_{ND} é descrita com dois conjuntos de efeitos não-determinísticos, tal que, $Add_1 = O(op) \cup Eff_d(op)$ e $Add_2 = O(op) \cup Eff_u(op)$. O conjunto Add_1 representa a operação executada com sucesso e produzindo os efeitos desejados e é obtido lendo o primeiro conjunto no escope de `oneof`. O conjunto Add_2 representa a operação executada com falha e é obtido lendo o segundo conjunto no escope de `oneof`.
- Os conjuntos de efeitos negativos Del_1 e Del_2 são deixados vazios, uma vez que Web services não produzem parâmetros de saída negativos (WANG et al., 2015; MARKOU; REFANIDIS, 2016). Neste sentido, sempre é possível salvar os parâmetros de saída produzidos ao longo da execução da composição a fim de que eles possam ser utilizados por diferentes operações.

Descrição do problema:

- Os parâmetros, pré-condições e efeitos de uma operação *op* se tornarão objetos do problema de planejamento cujos respectivos tipos são aqueles definidos no domínio.
- Os fatos que constituirão o estado inicial do mundo $s_0 \in S$ serão formados pelos parâmetros de entrada $I(r)$ da requisição r do usuário , tal que, $s_0 = I(r)$. Tais parâmetros serão identificadas pelos símbolos proposicionais `fato(x)`, onde `x` refere-se a um parâmetro em $I(r)$.
- O conjunto de objetivos G do agente de planejamento serão formados pelos parâmetros de saída $O(r)$ da requisição r do usuário,

tal que, $G = O(r)$. Tais parâmetros serão identificadas pelos símbolos proposicionais `fato(x)`, onde `x` refere-se a um parâmetro em $O(r)$.

A figura 6 ilustra o mapeamento entre os elementos envolvidos em uma composição de Web services para a linguagem NuPDDL segundo a convenção adotada neste trabalho. O mapeamento de um elemento a outro está explicitado por uma seta tracejada.

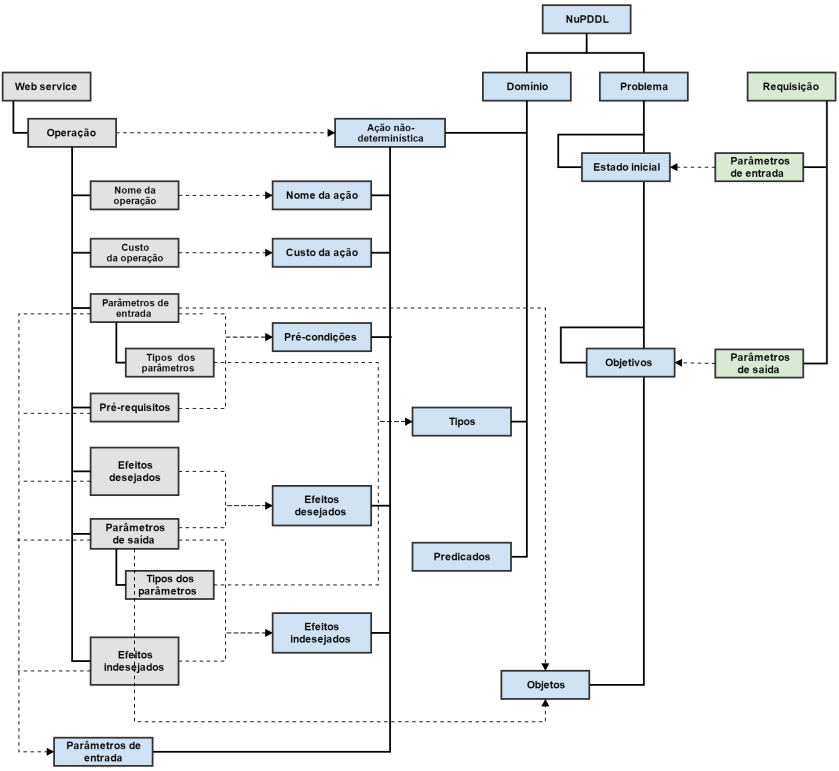

Figura 6 – Mapeamento de um problema de composição para a linguagem NuPDDL.

Mais especificamente, os efeitos indesejados mapeados para efeitos de uma ação não-determinística são somente aqueles descritos nas interfaces dos Web services (complementos dos efeitos desejados).

4.1.3 Módulo Mapeador para Planejamento Clássico

O arquivo NuPDDL resultante é enviado ao módulo *Mapeador para Planejamento Clássico* que o compila, extrai as informações necessárias para que o framework obtenha uma representação interna da tupla Σ_{ND} (ver definição 3.15) e, em seguida, obtém uma versão determinística Σ do problema de planejamento (ou seja, um problema de planejamento clássico).

Assim como (WINTERER; MATTMULLER; WEHRLE, 2015; MUISE; MCILRAITH; BELLE, 2014; KELLER; EYERICH, 2011; KUTER et al., 2008), este trabalho tira proveito das técnicas bem estudadas e conhecidas do planejamento clássico para buscar soluções para problemas de planejamento não-determinístico. O espaço de busca mapeado para sua versão determinística, por considerar um número maior de restrições (ver seção 3.1), tende a ser mais compacto doque o espaço de busca não-determinístico diminuindo a complexidade e o tempo de busca por um plano. Além do mais, a obter uma versão determinística do problema é necessária para viabilizar a execução da resolução através de SAT. O agente de planejamento pode buscar k planos determinísticos alternativos e combiná-los em um plano condicional para que, ao executar a solução resultante em um ambiente real, dinâmico e não-determinístico, o agente seja capaz de alcançar os seus objetivos.

Para que isso seja possível, a fonte do não-determinismo deve ser identificada e uma versão determinística deve ser obtida. Quando tal fonte for as ações, então para cada ação com n conjuntos de efeitos não-determinísticos derivam-se n ações determinísticas, onde cada uma dessas ações é instanciada com um daqueles n conjuntos de efeitos não-determinísticos. Neste sentido, o módulo Mapeador para Planejamento Clássico mapeia o conjunto de ações A_{ND} para um conjunto de ações determinísticas A, tal que, para cada ação $a_{ND} \in A_{ND}$ duas ações determinísticas $a', a'' \in A$ (ver definição 3.4) são obtidas, onde a', chamada de *ação de sucesso*, representa a operação correspondente executada com sucesso, e a'', chamada de *ação de falha*, representa a operação correspondente executada com falha.

Neste contexto, as ações determinísticas derivadas de uma mesma operação são chamadas de *ações contraparte*. Ações contraparte compartilham os mesmos pré-requisitos, mas diferenciam-se em seus conjuntos de efeitos determinísticos, assim $Pre(a') = Pre(a'') = I(op) \cup Pre(op)$, $Add(a') = O(op) \cup Eff_d(op)$ e $Add(a'') = O(op) \cup Eff_u(op)$. Ademais, $Del(a') = Del(a'')$ são deixados vazios. Como resultado, obtém-se um problema de planejamento determinístico Σ.

4.2.1 Módulos Tradutor para FNC, Solucionador SAT e Extrator de Planos

A fase de planejamento inicia-se pelo módulo *Tradutor para FNC* que, ao receber uma instância de um problema de planejamento determinístico Σ, o mapeia a uma fórmula lógica proposicional ψ na sua forma normal conjuntiva. Para isso, a função de mapeamento f (ver definição 3.8) e a definição de ação determinística foi estendida para suportar as características e definições de composições de Web services adotas neste trabalho. A definição de ação foi estendida apenas adicionando a ela o elemento $\mathcal{K}$(a), com $a \in A$, que representa o seu custo de execução.

O *axioma de estados sucessores* (ver definição 3.11) foi redefinido para o *axioma de estados sucessores contraídos*, onde excluiu-se o elemento *ActNotCausesL* já que não existem ações com efeitos negativos que neguem proposições, logo *ActNotCausesL* se torna um elemento vazio.

Definição 4.1. *(***Axioma de estados sucessores contraídos - AESC***) Um literal $p \in \mathbb{P}$ é verdadeiro no instante de tempo $t+1$ se e somente se:*

$$p_{i,t+1} \Leftrightarrow ActCausesL \vee p_{i,t}$$

onde, ActCausesL refere-se à disjunção entre os literais que representam as ações que têm em seus conjuntos de adição (Add) p (ou seja, tornam p verdadeiro no instante de tempo $t+1$).

Em segundo lugar, o axioma de objetivos (ver definição 3.13) foi redefinido para o *axioma de objetivos contraído*, onde omitem-se os literais em $\{\mathbb{P} - G\}$, pois, tanto negando-os quanto tornando-os positivos, nas circunstâncias deste trabalho, está incorreto. No primeiro caso, nenhum plano será encontrado, pois não existirão ações que neguem tais proposições. No segundo caso, obriga-se desnecessariamente que todas as proposições em $\mathbb{P}$ sejam verdadeiras (salvo quando $G = \mathbb{P}$), assim, se existir, o plano que soluciona o problema nestas circunstâncias tende a ser maior do que aquele que satisfaz unicamente o conjunto G.

Definição 4.2. *(***Axioma de objetivos contraído - AOC***) O conjunto de objetivos $G \subseteq \mathbb{P}$ do agente são mapeados da seguinte forma:*

$$\bigwedge\{p_{i,N} \mid p_{i,N} \in G\}$$

onde $p_{i,N}$ são os literais em $\mathbb{P}$ assumindo-se que os objetivos são alcançados no tempo $t = N$.

Os demais axiomas, APC e AEM, não necessitam ser modificados.

Definição 4.3. *Seja f' a função de mapeamento estendida da definição 3.8) que traduz instâncias de problemas de planejamento determinísticos Σ para SAT, considerando um* makespan *N, tal que $0 \leq t < N$, tal que:*

$$f'(\Sigma, N) = AEI_0 \wedge (\bigwedge_{0 \leq t < N} AESC_t \wedge APC_t \wedge AEM_t) \wedge AOC_N$$

onde, AEI, AESC, APC, AEM e AOE, referem-se aos axiomas de estado inicial, estados sucessores contraído*,* pré-condições*,* exclusão mútua *e* objetivos contraído*, respectivamente.*

O módulo *Tradutor para FNC* utiliza a função f' para obter uma fórmula lógica proposicional ψ em relação a um *makespan* N. Em seguida, a fórmula lógica proposicional ψ é transformada em uma forma normal conjuntiva (FNC) e enviada ao módulo *Solucionador SAT* que buscará por todos os modelos no intervalo (0,N] que a satisfaçam.

Ademais, apenas as ações de acerto são consideradas no mapeamento de planejamento para SAT, já que os efeitos produzidos pelas ações de falha não serão consumidos pelas demais ações (tais efeitos apenas informam que as operações que as originaram falharam em cumprir com as funções para as quais foram designadas). Neste sentido, os literais que representam os efeitos indesejados (produzidos pelas ações de falha) também não são considerados no mapeamento de planejamento para SAT. Como será visto em seguida, considerar apenas as ações de acerto é suficiente para que as corretas sequências de invocação e execução de operações de Web services sejam encontradas. As diferentes sequência de invocação e execução correspondem aos planos alternativos (composições alternativas de Web services). As ações de falha são consideradas apenas no momento em que a estratégia de contingência é construída servindo para indicar qual plano alternativo deve ser executado a partir de um estado de falha alcançado.

Como visto na seção 3.1.2 a busca por um modelo para problemas de planejamento mapeados a SAT é feita de modo iterativo.

Geralmente, o valor inicial de N é mínimo (isto é, igual a um, o que indica que um plano deve conter pelo menos uma ação). Tal valor é incrementado à medida que a fórmula lógica proposicional ψ resultante for insatisfazível. A busca procede até que o solucionador SAT determine que ψ é satisfazível ou que N alcance um valor máximo permitido de iterações (neste caso, se nenhum modelo for encontrado, então considera-se que não há solução para o problema de planejamento em relação ao *makespan* corrente). Se a fórmula for satisfazível, garante-se que o plano a ser extraído é de comprimento mínimo (GIUNCHIGLIA; MARATEA, 2007).

Entretanto, o modo tradicional de busca em planejamento SAT não é interessante para este trabalho, já que os modelos encontrados na iteração anterior serão novamente encontrados na iteração corrente assim como nas iterações futuras. Isto implica em um esforço computacional desnecessário, pois a cada nova iteração o solucionador SAT inicia a sua busca do zero. Deste modo, gasta-se desnecessariamente tempo para encontrar repetidamente um mesmo conjunto de modelos durante todas as iterações.

Neste sentido, este trabalho assume que a busca será feita para um *makespan* fixo cujo valor deve ser definido pelo usuário. Assim, o problema de planejamento é mapeado para uma instância de SAT, como indicado pela função de mapeamento estendida f' (ver definição 4.3). Deste modo, todos os modelos que satisfazem tal instância são buscados em uma única chamada ao SAT solver completo. O algoritmo 1 sumariza os passos adotados neste trabalho para a obtenção da fórmula ψ e do modelos σ que a satisfazem.

Caso o problema de planejamento seja satisfazível em relação ao *makespan* N, então todos os modelos σ são passadas ao módulo *Extrator de Planos* que, por sua vez, extraí um plano da fórmula lógica proposicional ψ para cada modelo obtido (onde k, o número total de planos, é igual ao número de modelos obtidos pelo solucionador SAT ou menor, caso exista a necessidade de que se aplique subsunção nos modelos). Isto é feito mediante análise dos modelos, onde um modelo determina, no máximo, para cada instante de tempo, através de um inteiro positivo (número DIMACS), um literal da fórmula lógica que se refere a uma ação como semanticamente verdadeiro. As ações são recuperadas e ordenadas de modo crescente de acordo com os seus instantes de tempo.

Algorithm 1 Obtendo ψ e os modelos σ

```
1: procedure PLANEJAMENTO_PARA_SAT(w, N)
2:     ψ ← ∅
3:     for i ← 0,..., N-1 do
4:         ψ ← ψ ∧ AESC_i ∧ APC_i ∧ AEM_i
5:     end for
6:     ψ ← AEI_0 ∧ γ ∧ AOC_N
7:     ψ ← Traduzir_Para_FNC(ψ)
8:     if Invocar_solucionador_sat_completo(ψ) = satisfazível then
9:         return todos os modelos σ
10:    else
11:        print "problema insatisfazível"
12:    end if
13: end procedure
```

4.2.2 Módulo Fusor de Planos

Os k planos obtidos são enviados ao módulo *Fusor de planos* que constrói a estratégia de contingência na forma de uma árvore de decisão binária como a ilustrada na figura 7 (apesar da figura 7 se encontrar distante desta parte do texto, olhar tal ilustração para ter uma visão geral e mais concreta de uma árvore binária no contexto deste trabalho permite entender melhor a descrição da mesma). Na árvore binária, os nodos e as arestas representam, respectivamente, estados do mundo (que refletem os estados da execução de uma composição de Web services) e transições entre estados causadas pelas execuções das ações (execuções das operações). A execução de uma ação de acerto é representada pela aresta da esquerda que se conecta a um *nodo de acerto* (estado no qual efeitos desejados são alcançados), ao passo que, a execução de uma ação de falha é representada pela aresta da direita que se conecta a um *nodo de falha* (estado no qual efeitos indesejáveis são alcançados).

A raiz representa o estado inicial $s_0 \in S$ do mundo e uma folha pode representar um nodo de aceitação (um estado $s_g \in S_g$) ou um nodo morto (estado a partir do qual não há caminho para um nodo de aceitação). Um caminho que parte da raiz para um nodo de aceitação corresponde a uma execução com sucesso da estratégia de contingência, isto é uma composição de execução de operações de Web services que satisfaz os objetivos do usuário. Por outro lado, um caminho que parte da raiz para um nodo morto corresponde a uma execução da estratégia

de contingência com insucesso.

O algoritmo 2 apresenta os passos necessários para a construção da estratégia de contingência. O primeiro plano (ρ no algoritmo 2) a ser adicionado na árvore é aquele cujo comprimento (número de ações) e custo (soma do custo de execução $\mathcal{K}(a)$ de cada ação a que compõem o plano ρ) são mínimos para alcançar os objetivos do usuário. Cada nodo tem na sua estrutura uma aresta à direita e outra à esquerda (ponteiros para os nodos de acerto e falha, respectivamente), flags que indicam quais ações (de acerto e de falha) são executadas sobre o nodo, e a sequência (*seq*) de ações que alcançam o nodo corrente, ou seja, aquelas executadas a partir da raiz até o pai de tal nodo.

Algorithm 2 Construindo a Estratégia de Contingência.

```
procedure CONSTRUIR_ESTRATÉGIA(Plano ρ, Sequência seq)
    Nodo n = new Nodo();
    while ρ ≠ null and (n.getArestaEsquerda()=null or
        n.getArestaDireita()=null) do
        Ação a = ρ.getAção();
        n.setSequência(seq);
        n.setAçãoDeAcerto(a);
        n.setArestaEsquerda(Construir_Estratégia(ρ,seq ∧ a));
        Ação af = getAçãoContraparte(a);
        n.setAçãoDeFalha(af);
        ρ' = getPlanoAlternativo(ρ, seq ∧ af);
        n.setArestaDireita(Construir_Estratégia(ρ',seq ∧ af));
    end while
    return n
end procedure
```

As ações são recursivamente obtidas pelo método *getAção*, removidas do plano ρ corrente, e inseridas uma a uma na árvore binária. A inserção do plano ρ na árvore é finalizada quando tal plano se torna nulo (sem ações). Após retornar de uma chamada recursiva, o algoritmo 4.2.2 insere na árvore a ação de falha correspondente (contraparte af) à ação de acerto já adicionada no nodo corrente. Para cada nodo de falha alcançado, o método *getPlanoAlternativo* é chamado para obter um plano alternativo a ser inserido a partir dele. Se não existe plano alternativo para tal nodo, então o nodo de falha se torna um nodo morto.

O método *getPlanoAlternativo* retorna o melhor plano alternativo, onde tal plano: i) possui a maior sequência de ações em comum

com aquelas contidas no caminho $seq \wedge fa$; ii) possui o menor comprimento; iii) possui o menor custo; e iv) não possui ações de acerto que são contraparte de ações de falha entre a raiz e o nodo de falha a partir do qual o plano será inserido. A sequência em comum é removida do plano alternativo, já que não há necessidade de reexecutar as ações naquele caminho novamente, pois os parâmetros produzidos até então podem ser armazenados e reutilizados. A construção da árvore encerra quando tentou-se adicionar algum plano em cada nodo de falha (pode ocorrer de existirem planos que não puderam ser inseridos na árvore binária). Ao final, a raiz da árvore é retornada e passada ao módulo *Tradutor para Processo Executável*.

Para simular a execução da árvore, o agente de planejamento deve atuar como um orquestrador invocando a operação que originou as ações que partem de um dado nodo. Ao receber a mensagem de resposta, o agente analisa o seu conteúdo e determina se a operação invocada gerou os efeitos desejados. Em caso positivo, o agente prossegue a execução para o filho à esquerda daquele nodo. Em caso negativo, então, o agente prossegue a execução para o filho à direita no qual, se possível, um plano alternativo foi inserido. Se no nodo de falha alcançado não houver um plano para ser executado, então o agente realiza *backtracking* na árvore a fim de encontrar um plano ainda não executado. É permitido ao agente retornar à raiz da árvore (isto é, reiniciar a sua execução), em no máximo n vezes (valor que deve ser configurado pelo usuário), quando não existirem mais planos a serem tentados pelo agente de planejamento.

Assim, aumenta-se as chances de que os objetivos do usuário sejam satisfeitos em ambientes dinâmicos e não-determinísticos, onde problemas de execução são propícios a acontecer e as operações falharem em produzir os efeitos desejados.

4.3 FASE DE EXECUÇÃO

4.3.1 Módulo Tradutor para Processo Executável

Na fase de execução, o módulo *Tradutor para Processo Executável* converte a árvore binária em um processo global executável de acordo com um código de descrição de processos (como BPEL ou BPMN, por exemplo). Para isso, realiza-se uma leitura em ordem de modo recursivo sobre a árvore de decisão binária transcrevendo os planos nela fundidos em sequências de invocações de operações, onde cada plano equivale a

um processo. Como resultado, ao final da leitura, obtém-se um código que funde os processos e indica qual executar quando as operações falham em obter os efeitos desejados. Em outras palavras, executar o código do processo global equivale à forma como a árvore binária é executada.

A medida que as chamadas recursivas são realizadas, a operação do Web service que originou as ações (de acerto e falha) executadas sobre o nodo que está sendo visitado é recuperada. Para cada operação recuperada, uma estrutura de controle do tipo *if-then-else* é inserida no código resultante a fim de determinar se, em tempo de execução, os efeitos desejados foram produzidos por aquela operação. A operação subsequente é adicionada no escopo do *then*, e no retorno da chamada recursiva o plano alternativo indicado na árvore binária para ser executado quando a operação corrente falhar é inserido ao escopo do *else*.

Se tal plano não existir, então é necessário buscar por algum outro caminho promissor na árvore a alcançar os objetivos do usuário. Este caminho é naturalmente obtido pela execução da leitura em ordem que, ao visitar uma folha, retorna para o primeiro nodo que possui subárvore à direita ainda não visitada. Caso a raiz de tal subárvore possua filho à esquerda (o que significa que aquela subárvore não é constituída de apenas uma folha e nela existe, pelo menos, um caminho a ser percorrido até um estado de objetivos), então um plano alternativo é adicionado no escopo do *else* corrente (como poderá ser conferido na figura 7 do exemplo da seção 4.4 quando o nodo s_6 é alcançado). Caso não exista caminho alternativo para ser percorrido naquela subárvore, então, nenhum código é adicionado ao *else* corrente e, sob as mesmas condições, a leitura em ordem retorna para o próximo nodo com subárvore à direita também não visitada.

Caso, durante a leitura em ordem, detecte-se que não existe plano alternativo para ser adicionado ao *else* correte, então retorna-se à raiz da árvore a partir da qual a leitura se reinicia. Deste modo, a árvore binária é adicionada, na mesma dinâmica anteriormente descrita, aquele *else*. A leitura em ordem retorna à raiz da árvore binária n vezes (valor que deve ser configurado pelo usuário), uma vez para cada *else* que não possui plano alternativo a ser inserido a partir dele. Este procedimento simula o retorno à raiz da árvore para reiniciar a sua execução quando não existem mais planos a serem tentados pelo agente de planejamento (como ocorre quando os nodos s_{26}, s_{27} e s_{28} da figura 7 são alcançados).

Segundo a ótica do planejamento não-determinístico, a árvore binária obtida neste trabalho e o processo global são classificadas como soluções forte cíclicas, ou seja, garantem que para todo estado do mundo

é possível alcançar um estado de objetivos, entretanto paga-se o preço de que ciclos sejam executados até que um conjunto de efeitos desejados sejam obtidos (mas não há garantia de que tal conjunto seja efetivamente obtido). Ademais, o código resultante deve ser descrito com o suporte de alguma linguagem de execução de processos como a BPEL4WS (*Business Process Execution Language for Web Services*) ou BPMN (*Business Process Model and Notation*), por exemplo.

4.3.2 Módulo Motor de Execução

O código do processo global é passado ao módulo *Motor de Execução* que o executa e acompanha o seu progresso a fim de detectar quando uma operação falha em produzir os efeitos desejados ou excede o tempo de execução permitido (*timeout* configurado pelo usuário). Ao determinar que um plano alternativo deve ser executado, o motor deve providenciar meios para reverter (*rollback*) os efeitos produzidos pela operação que falhou. Qualquer motor de execução de processos pode ser usado para desempenhar as funções do módulo Motor de Execução. Exemplos de motores são: Express BPEL (CodeBrew Technologies, 2012), Apache ODE (Apache Software Foundation, 2013) e Open ESB (OpenESB Community, 2013).

Na seção seguinte será ilustrado um exemplo de como uma composição resiliente de Web services semânticos é obtida de acordo com a proposta deste trabalho.

4.4 ESTUDO DE CASO: OBTENDO UMA COMPOSIÇÃO RESILIENTE DE WEB SERVICES SEMÂNTICOS

Nesta seção, apresenta-se um exemplo concreto de como uma composição resiliente de Web services semânticos é obtida segundo as fases do framework proposto neste capítulo.

Exemplo 7. *Considere o exemplo 6 (pág. 67) no qual um usuário requisita um serviço para comprar passagens aéreas para viajar entre duas cidades e que não exista Web service em W com operação capaz de satisfazer o seu objetivo. Suponha que a solução para este problema será alcançada através do framework apresentado nas seções anteriores. Deste modo, o módulo Motor de Descoberta obterá um conjunto de Web services $W' \subseteq W$ com operações que satisfazem parcialmente a requisição do usuário e suponha, também, que para cada operação em W'*

existe um, e apenas um, Web service com operação equivalente (operação alternativa) com os mesmos conjuntos de parâmetros de entrada e saída.

As operações em W' são reescritas para que os parâmetros que possuem tipos semanticamente equivalentes passem a ser identificados pelo mesmo nome de parâmetro e mesmo nome de tipo. Assim, os parâmetros semanticamente equivalentes `voosDisponíveis` e `voosComAssentoLivres` das operações `ListarVoosDisponíveis (LVDop)` e `ReeservarVoosDisponíveis (RVDop)`, respectivamente, passam a ser identificados unicamente pelo nome de pârametro `voosDisponíveis` e pelo nome de tipo `VoosDisponíveis`. Os parâmetros `preço` e `valor` das operações `ReeservarVoosDisponíveis` e `PagamentoViaCrédito (PVCop)`, respectivamente, passam a ser identificados unicamente pelo nome de parâmetro `preço` e pelo nome de tipo `Preço`. Em seguida, o problema de composição de Web services é descrito em NuPDDL pelo módulo Tradutor para NuPDDL levando em consideração o conjunto W' e a requisição r. A descrição em NuPDDL resultante é a seguinte[2]:

```
(define (domain EXEMPLO7)
(:types DataPartida
        DataRetorno
        AeroportoOrigem
        AeroportoDestino
        Classe
        VoosDisponíveis
        Preço
        PreçoMáximo
        VoosReservados
        Transação
        Condição_1 ; Refere-se ao efeito desejado da operação LVDop
                   ; e a pré-condição da operação RVDop
        Condição_2 ; Refere-se à pré-condição da operação PVCop
        Condição_3 ; Refere-se ao efeito desejado da operação RVDop
        Condição_4
        Condição_5 ; As condições 4 e 5 referem-se ao conjunto
                   ; de efeitos desejados da operação PVCop
```

[2]Por questões de simplificação as operações alternativas não estão descritas no arquivo NuPDDL resultante.

```
        Condição_6 ; Refere-se ao efeito indesejado da operação LVDop
        Condição_7 ; Refere-se ao efeito indesejado da operação RVDop
        Condição_8
        Condição_9 - Fato ; As condições 8 e 9 referem-se aos efeitos
                  ; indesejados da operação PVCop
                  ; Todas as condições declaradas são subtipos de Fato
(:predicates (fato ?x - Fato))
(:functions  (custo))
(:action ListarVoosDisponíveis ; LVDop
      (:parameters (?x - DataPartida
                    ?y - DataRetorno
                    ?z - AeroportoOrigem
                    ?p - AeroportoDestino
                    ?q - Classe
                    ?r - VoosDisponíveis
                    ?s - Condição_1
                    ?t - Condição_6))
      (:precondition (and (fato ?x)
                          (fato ?y)
                          (fato ?z)
                          (fato ?p)
                          (fato ?q)))
      (:effect (oneof (and (fato ?r) ; Derivará Add_1
                           (fato ?s)
                           (assign (custo) 1))
                      (and (fato ?r) Derivará Add_2
                           (fato ?t)
                           (assign (custo) 1)))))
(:action ReservarVoosDisponíveis ; RVDop
      (:parameters (?x - VoosDisponíveis
                    ?y - PreçoMáximo
                    ?z - VoosReservados
                    ?p - Preço
                    ?q - Condição_1
                    ?r - Condição_3
                    ?s - Condição_7))
      (:precondition (fato ?x)
                     (fato ?y)
                     (fato ?q)
```

```
      (:effect (oneof (and (fato ?z) ;Derivará Add_1
                           (fato ?p))
                           (fato ?r)
                           (assign (custo) 1))
                      (and (fato ?z) ;Derivará Add_2
                           (fato ?p)
                           (fato ?s)
                           (assign (custo) 1)))))
(:action PagamentoViaCrédito ; PVCop
      (:parameters (?x - CartãoDeCrédito
                   ?y - Preço
                   ?z - Transação
                   ?p - Condição_2
                   ?q - Condição_4
                   ?r - Condição_5
                   ?s - Condição_8))
                   ?t - Condição_9))
      (:precondition (fato ?x)
                     (fato ?y)
                     (fato ?p))
      (:effect (oneof (and (fato ?q) ;Derivará Add_1
                           (fato ?r)
                           (fato ?z)
                           (assign (custo) 1))
                      (and (fato ?s) ;Derivará Add_2
                           (fato ?t)
                           (fato ?z)
                           (assign (custo) 1))))))
(define (problem INSTÂNCIA_EXEMPLO7)
(:domain EXEMPLO7)
(:objects (dataPartida - DataPartida)
          (dataRetorno - DataRetorno)
          (aeroportoOrigem - AeroportoOrigem)
          (aeroportoDestino - AeroportoDestino)
          (classe - Classe)
          (voosDisponíveis - VoosDisponíveis)
          (voosReservados - VoosReservados)
          (preçoMáximo - PreçoMáximo)
          (preço - Preço)
```

```
            (cartãoDeCrédito - CartãoDeCrédito)
            (transação - Transação)
            (voosDisponíveis.tamanho>0 - Condição_1)
            (cartaoDeCrédito.saldo≥valor - Condição_2)
            (voosReservados.tamanho>0 - Condição_3)
            (transação.estado=aprovado - Condição_4)
            (cartaoDeCrédito.saldo=cartaoDeCrédito.saldo−
             valor - Condição_5)
            (voosDisponíveis.tamanho≤0 - Condição_6)
            (voosReservados.tamanho≤0 - Condição_7)
            (transação.estado≠aprovado - Condição_8)
            (cartaoDeCrédito.saldo≠cartaoDeCrédito.saldo−
             valor - Condição_9)
 (:init (fato dataRetorno)
        (fato dataPartida)
        (fato aeroportoOrigem)
        (fato aeroportoDestino)
        (fato classe)
        (fato preçoMáximo)
        (fato cartãoDeCrédito)
        (fato voosReservados.tamanho>0)
        (fato cartaoDeCrédito.saldo≥valor)
        (= (custo) 0)) ; Inicializando o valor de custo com zero
 (:goal (fato transação)))
```

O módulo Mapeador para Planejamento Clássico recebe a descrição do problema de planejamento em NuPDDL, o compila e extrai as informações necessárias para obter uma representação interna da tupla Σ_{ND}. O conjunto de símbolos proposicionais $\mathbb{P}$ é construído instanciando o predicado `fato` com os objetos definidos na parte `problem` do arquivo NuPDDL. Tais símbolos são apresentados a seguir e estão descritos na forma p_i, onde p refere-se a um símbolo proposicional e $i \in \mathbb{N}^+$.

Em seguida o módulo Mapeador para Planejamento Clássico obtém, a partir do conjunto de ações não-determinísticas A_{ND}, o conjunto de ações determinísticas A, tal que, para cada ação $a_{ND} \in A_{ND}$ duas ações determinísticas a' (ação de acerto) e a'' (ação de falha) são obtidas. As ações obtidas são as seguintes[3]:

[3]As operações de Web services alternativas estão identificadas com o pós-fixo 2 ao final de seus nomes. As ações derivadas de acerto são identificadas por índices

p_1 = fato(dataPartida)
p_2 = fato(dataRetorno)
p_3 = fato(aeroportoOrigem)
p_4 = fato(aeroportoDestino)
p_5 = fato(classe)
p_6 = fato(voosDisponíveis)
p_7 = fato(voosReservados)
p_8 = fato(preço)
p_9 = fato(preçoMáximo)
p_{10} = fato(cartãoDeCrédito)
p_{11} = fato(transação)
p_{12} = fato(voosDisponíveis.tamanho>0)
p_{13} = fato(voosReservados.tamanho>0)
p_{14} = fato(transação.estado=aprovado)
p_{15} = fato(cartaoDeCrédito.saldo=cartaoDeCrédito.saldo−valor)
p_{16} = fato(cartaoDeCrédito.saldo≥valor
p_{17} = fato(voosDisponíveis.tamanho≤0)
p_{18} = fato(voosReservados.tamanho≤0)
p_{19} = fato(transação.estado≠aprovado)
p_{20} = fato((cartaoDeCrédito.saldo≠cartaoDeCrédito.saldo−valor)

`ListarVoosDisponíveis` deriva as ações a_1 e a_2 com $Pre(a_1) = Pre(a_2) = \{p_1,p_2,p_3,p_4,p_5\}$, $Add(a_1) = \{p_6,p_{12}\}$, $Add(a_2) = \{p_6,p_{17}\}$ e $\mathcal{K}(a_1) = \mathcal{K}(a_2) = 1$. O mesmo ocorre para `ListarVoosDisponiveis2 (LVDop2)` que deriva as ações a_3 e a_4, onde os seus elementos são os mesmos elementos das ações a_1 e a_2, respectivamente, com exceção de $\mathcal{K}(a_3) = \mathcal{K}(a_4) = 3$. `ReservarVoosDisponiveis` deriva as ações a_5 e a_6 com $Pre(a_5, a_6) = \{p_6,p_8,p_{12}\}$, $Add(a_5) = \{p_7,p_8,p_{13}\}$, $Add(a_6) = \{p_7,p_8,p_{18}\}$ e $\mathcal{K}(a_5, a_6) = 1$. O mesmo ocorre para `ReservarVoosDisponiveis2(RVDop2)` que deriva as ações a_7 e a_8, onde os seus elementos são os mesmos de a_5 e a_6, respectivamente, com exceção de $\mathcal{K}(a_7, a_8) = 2$. `pagamentoViaCrédito` deriva as ações a_9 e a_{10} com $Pre(a_9) = Pre(a_{10}) = \{p_8,p_{10},p_{16}\}$, $Add(a_9) = \{p_{11},p_{14},p_{15}\}$, $Add(a_{10}) = \{p_{11},p_{19},p_{20}\}$ e $\mathcal{K}(a_9) = \mathcal{K}(a_{10}) = 2$. Finalmente, `pagamentoViaCrédito2 (PVCop2)` deriva as ações a_{11} e a_{12}, onde os seus elementos são os mesmos elementos das ações a_9 e a_{10}, respectivamente, com exceção de $\mathcal{K}(a_{11}) = \mathcal{K}(a_{12}) = 3$.

Levando em consideração o conjunto de ações A, o estado inicial $s_0 \in S$ e o conjunto de objetivos G, o módulo Tradutor para FNC obtém uma fórmula lógica proposicional ψ de acordo com a definição 4.3 e, em

ímpares, ao passo que as ações de falha são identificadas por índices pares.

seguida, a transforma na sua forma normal conjuntiva. Considerando o *makespan* $N = 3$ (existem três ações de acerto distintas que podem ser combinadas em uma determinada sequência formando planos de comprimento três), os axiomas obtidos são os seguintes[4].

Estado inicial e objetivos:

$$\begin{aligned} AEI_0 = &\quad p_{1,0} \wedge p_{2,0} \wedge p_{3,0} \wedge p_{4,0} \wedge p_{5,0} \wedge p_{9,0} \wedge p_{10,0} \\ AO_6 = &\quad p_{11,6} \end{aligned}$$

Axiomas de exclusão mútua:

$$\begin{aligned} &(\neg a_{1,0} \vee \neg a_{3,0}) \wedge (\neg a_{1,0} \vee \neg a_{5,0}) \wedge \ldots \wedge (\neg a_{1,0} \vee \neg a_{11,0}) \\ &(\neg a_{3,0} \vee \neg a_{5,0}) \wedge (\neg a_{3,0} \vee \neg a_{7,0}) \wedge \ldots \wedge (\neg a_{3,0} \vee \neg a_{11,0}) \\ &(\neg a_{5,0} \vee \neg a_{7,0}) \wedge (\neg a_{5,0} \vee \neg a_{9,0}) \wedge \ldots \wedge (\neg a_{5,0} \vee \neg a_{11,0}) \\ &(\neg a_{7,0} \vee \neg a_{9,0}) \wedge (\neg a_{7,0} \vee \neg a_{11,0}) \\ &(\neg a_{9,0} \vee \neg a_{11,0}) \end{aligned}$$

Axiomas de pré-condições:

$$\begin{aligned} a_{1,0} &\rightarrow p_{1,0} \wedge p_{2,0} \wedge p_{3,0} \wedge p_{4,0} \wedge p_{5,0} \\ a_{3,0} &\rightarrow p_{1,0} \wedge p_{2,0} \wedge p_{3,0} \wedge p_{4,0} \wedge p_{5,0} \\ a_{5,0} &\rightarrow p_{6,0} \wedge p_{9,0} \wedge p_{12,0} \\ a_{7,0} &\rightarrow p_{6,0} \wedge p_{9,0} \wedge p_{12,0} \\ a_{9,0} &\rightarrow p_{8,0} \wedge p_{10,0} \wedge p_{16,0} \\ a_{11,0} &\rightarrow p_{8,0} \wedge p_{10,0} \wedge p_{16,0} \end{aligned}$$

Axiomas de estados sucessores contraído:

$$\begin{aligned} &p_{1,1} \leftrightarrow p_{1,0} && p_{2,1} \leftrightarrow p_{2,0} \\ &p_{3,1} \leftrightarrow p_{3,0} && p_{4,1} \leftrightarrow p_{4,0} \\ &p_{5,1} \leftrightarrow p_{5,0} && p_{6,1} \leftrightarrow a_{1,0} \vee a_{3,0} \vee p_{6,0} \\ &p_{7,1} \leftrightarrow a_{5,0} \vee a_{7,0} \vee p_{7,0} && p_{8,1} \leftrightarrow a_{5,0} \vee a_{7,0} \vee p_{8,0} \\ &p_{9,1} \leftrightarrow p_{9,0} && p_{10,1} \leftrightarrow p_{10,0} \\ &p_{11,1} \leftrightarrow a_{9,0} \vee a_{11,0} \vee p_{11,0} && p_{12,1} \leftrightarrow a_{1,0} \vee a_{3,0} \vee p_{12,0} \\ &p_{13,1} \leftrightarrow a_{5,0} \vee a_{7,0} \vee p_{13,0} && p_{14,1} \leftrightarrow a_{9,0} \vee a_{11,0} \vee p_{14,0} \\ &p_{15,1} \leftrightarrow a_{9,0} \vee a_{11,0} \vee p_{15,0} && p_{16,1} \leftrightarrow a_{9,0} \vee a_{11,0} \vee p_{16,0} \end{aligned}$$

Ao receber ψ, o módulo Solucionador SAT busca por todos os modelos σ que satisfazem a fórmula lógica. Para cada modelo ψ encon-

[4]Os axiomas de estado sucessores contraído, exclusão mútua e pré-condições foram codificados apenas para o instante de tempo 0 por questões de simplicidade.

trado, o módulo Extrator de Planos extraí um plano de ψ. Os seguintes planos ρ, ordenados de forma crescente, primeiramente, pelo tamanho $l(\rho)$ e, em segundo lugar, pelo custo $\mathcal{K}(\rho)$, podem ser extraídos de ψ:

	ρ	$l(\rho)$	$\mathcal{K}(\rho)$
1 :	$a_{1,0} \wedge a_{5,1} \wedge a_{9,2}$	3	4
2 :	$a_{1,0} \wedge a_{5,1} \wedge a_{11,2}$	3	5
3 :	$a_{1,0} \wedge a_{7,1} \wedge a_{9,2}$	3	5
4 :	$a_{3,0} \wedge a_{5,1} \wedge a_{9,2}$	3	6
5 :	$a_{1,0} \wedge a_{7,1} \wedge a_{11,2}$	3	6
6 :	$a_{3,0} \wedge a_{5,1} \wedge a_{11,2}$	3	7
7 :	$a_{3,0} \wedge a_{7,1} \wedge a_{9,2}$	3	7
8 :	$a_{3,0} \wedge a_{7,1} \wedge a_{11,2}$	3	8

Em seguida, os planos são enviados ao módulo *Fusor de planos* que irá construir a estratégia de contingência, ilustrada pela figura 7, onde as arestas em vermelho representam os caminhos promissores que o agente de planejamento pode tentar após alcançar um nodo morto.

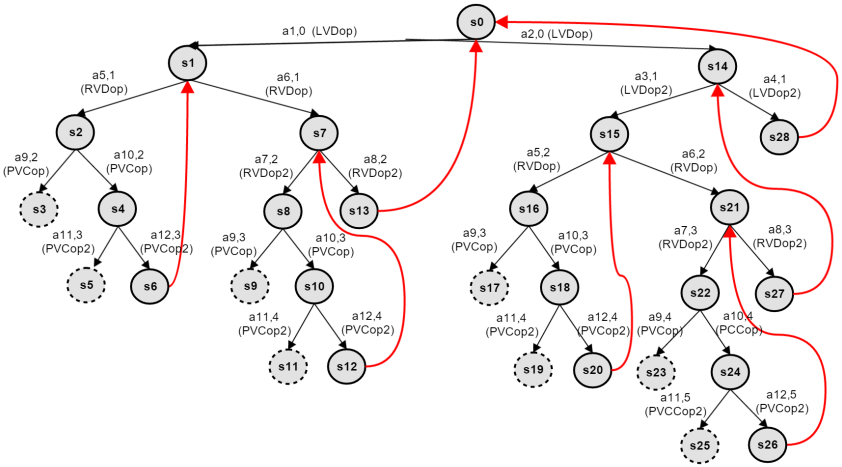

Figura 7 – Exemplo de uma estratégia de contingência.

A árvore binária começa a ser construída a partir da inserção do plano ρ_1 no nodo s_0 (raiz), pois, dentre todos os planos encontrados, ρ_1 é o plano com menor comprimento e custo. A inserção de tal plano na árvore cria os nodos de falha s_4 e s_7. Primeiramente, tenta-se encontrar

um plano alternativo para ser inserido a partir do nodo s_4. Analisando os planos restantes (de ρ_2 a ρ_8), o melhor plano a ser inserido é o plano ρ_2 porque, de acordo com os critérios de inserção de planos na árvore (ver seção 4.2.2), este plano possui o maior número de ações em comum com aquelas presentes no caminho que parte da raiz até o nodo s_4 (enquanto que o plano ρ_3 possui apenas uma ação em comum para o mesmo caminho, e os demais planos possuem zero ações em comum), menor custo e tamanho. Após a escolha, as ações em comum são removidas de ρ_2.

A adição da ação $a_{11,3}$ no nodo de falha s_4 torna o plano ρ_2 nulo, então o algoritmo 2 interrompe a geração do caminho atual na árvore e o nodo s_5 se torna um nodo de aceitação. Como não existe plano alternativo que cumpra os critérios de inserção para o nodo de falha s_6 (os demais planos não atendem o critério de inserção *iv*), então este nodo passa a ser considerado um nodo morto. Os caminhos restantes da árvore são construídos nesta mesma dinâmica.

A árvore resultante é passada ao módulo *Tradutor para Processo Executável* que a converte em um processo global executável. O seguinte pseudo-código reflete como parte dos planos sobrepostos na árvore binária (planos ρ_1 e ρ_2) serão invocados pelo motor quando, ao executarem em ambientes dinâmicos e não-determinísticos, uma de suas operações falhar em cumprir as funcionalidades para as quais foi designada:

```
invocar LVDop com dataPartida, dataRetono, aeroportoOrigem, aeroportoDestino, classe
if retorno voosDisponíveis.lista ≠ ∅
    invocar RVDop com voosDisponíveis, preçoMáximo
        if retorno voosReservados.tamanho > 0
        invocar PVCop com cartãoDeCrédito, preço
            if retorno transação.estado = aprovado ∧
               cartaoDeCédito.saldo = cartaoDeCrédito.saldo−preço
                   encerrar processo com sucesso
            else
                invocar PVCop2 com cartãoDeCrédito, preço
                if retorno transação.estado = aprovado ∧
                   cartaoDeCédito.saldo = cartaoDeCrédito.saldo−preço
                       encerrar processo com sucesso
                else
                    invocar RVDop2 com voosDisponíveis, preçoMáximo
                    if retorno voosDisponíveis.tamanho> 0
                    ...
        else
        ...
else
   invocar LVDop2
   if voosDisponíveis.lista ≠ ∅
   ...
```

No capítulo seguinte serão apresentadas as informações sobre a implementação da proposta deste trabalho, assim como os testes reali-

zados e discussão dos resultados obtidos.

5 IMPLEMENTAÇÃO, TESTES E RESULTADOS

Este capítulo descreve, inicialmente, o desenvolvimento de um protótipo que implementa parcialmente os módulos da arquitetura do framework proposto nesta dissertação. Em seguida, os testes realizados e resultados preliminarmente obtidos são discutidos. A saber, os módulos implementados foram: Mapeador para Planejamento Clássico, Solucionador SAT, Decodificador de FNC e Fusor de Planos. Os módulos Motor de Descoberta, Tradutor para NuPDDL, Tradutor para Processo Executável e Motor de Execução serão implementados em uma versão futura deste trabalho. Ademais, os módulos foram implementados na linguagem Java 8.

5.1 MÓDULO MAPEADOR PARA PLANEJAMENTO CLÁSSICO

O módulo *Mapeador para Planejamento Clássico* é responsável por compilar o arquivo NuPDDL que recebe e extrair dele um conjunto de informações necessárias a fim de obter uma representação interna de uma instância de planejamento não-determinística Σ_{ND} para, em seguida, obter uma versão determinística Σ (problema de planejamento clássico) deste mesmo problema.

A figura 8 ilustra o conjunto de submódulos implementados para desempenhar as funções do módulo Mapeador para Planejamento Clássico:

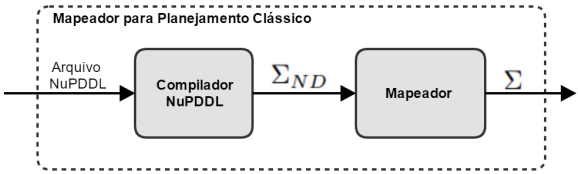

Figura 8 – Submódulos do Mapeador para Planejamento Clássico.

O submódulo Compilador NuPDDL implementa um compilador cujo desenvolvimento se deu através do gerador automático de analisadores léxicos e sintáticos JavaCC (COPELAND, 2007). Ao receber a especificação de uma gramática livre de contexto, JavaCC gera um programa escrito na linguagem Java capaz de determinar se um fluxo de

caracteres de entrada é uma sentença válida de uma dada linguagem através das análises léxica e sintática.

A gramática utilizada para tal fim foi a gramática que descreve problemas de planejamento não-determinístico apresentada na seção 3.1.3, a gramática NuPDDL. O fluxo de caracteres de entrada, sobre o qual o compilador opera, refere-se a um arquivo de texto no qual o usuário descreve um problema de composição de Web services como um problema de planejamento não-determinístico. A fase de análise semântica foi desenvolvida manualmente, uma vez que o JavaCC, assim como os demais geradores automáticos de analisadores léxicos e sintáticos, não implementa automaticamente esta fase. Durante a análise semântica, verifica-se através da aplicação de um conjunto de regras semânticas se as sentenças declaradas pelo usuário possuem significados coerentes. Para este trabalho, a análise semântica foi implementada de acordo com a técnica de tradução dirigida pela sintaxe (AHO; SETHI; LAM, 2008).

Esta técnica determina que ações semânticas sejam associadas às regras da gramática. À medida que as ações semânticas são aplicadas, símbolos proposicionais, ações não-determinísticas, estado inicial e objetivos declarados no arquivo de entrada são armazenados em diferentes tabelas de símbolos, uma tabela para cada um destes elementos. Estas tabelas guardam todas as informações relevantes para que o problema de planejamento descrito em tal arquivo seja reduzido a uma instância de SAT. Além do mais, um símbolo proposicional representa um fato em um determinado instante de tempo. Deste modo, um instante de tempo é atribuído a um símbolo proposicional ao ser instanciado. A semântica do fato no instante de tempo atribuído ao símbolo proposicional será posteriormente determinada pelo solucionador SAT.

As ações semânticas que foram implementadas verificam se: os objetos do mundo possuem tipos válidos; os símbolos proposicionais possuem nomes válidos, número e tipo corretos de parâmetros; estado inicial e objetivos são formados por símbolos proposicionais existentes; parâmetro das ações possuem tipos válidos.

Símbolos proposicionais e ações não-determinísticas são instanciados com um instante de tempo. Tal atributo é um inteiro que indica o momento para o qual a ação foi criada para ser executada ou para o qual a proposição foi criada para ser semanticamente verdadeira. Entretanto, o solucionador SAT irá determinar se tal ação ou símbolo proposicional poderá, respectivamente, ser executada ou ser verdadeiro naquele instante. Inicialmente, o instante de tempo é zero e diferentes valores são atribuídos a novas ações e símbolos proposicionais que são

criados no momento em que o problema de planejamento está sendo mapeado a SAT.

Se o processo de compilação for bem sucedido, então o submódulo Mapeador é invocado para mapear o conjunto de ações não-determinística A_{ND} em um conjunto de ações determinísticas A, conforme descrito na seção 4.1, obtendo, assim, um problema de planejamento determinístico Σ. Em seguida, Σ é passado ao módulo Tradutor para FNC.

5.2 MÓDULO TRADUTOR PARA FNC

O módulo Tradutor para FNC é responsável por obter uma fórmula lógica proposicional ψ de acordo com a definição 4.3 e transformá-la na sua forma normal conjuntiva. A figura 9 ilustra o conjunto de submódulos implementados para desempenhar a função do módulo Tradutor para FNC:

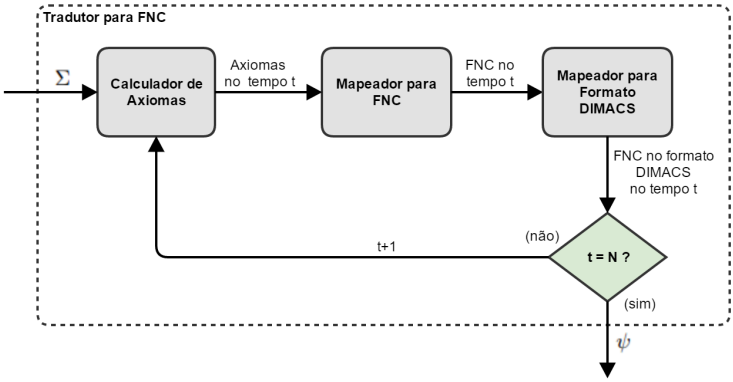

Figura 9 – Submódulos do Tradutor para FNC.

O submódulo *Calculador de axiomas* implementa um conjunto de métodos que calculam os axiomas que compõem a função estendida f' (ver definição 4.3) utilizada no mapeamento de instâncias de planejamento determinísticos a instâncias de SAT. Os axiomas são calculados individualmente para um instante de tempo $t \in [0, N)$ e conectados uns aos outros por um e lógico. A fórmula lógica resultante é enviada ao submódulo *Mapeador para FNC* que a mapeia para a sua forma normal

conjuntiva correspondente.

Em seguida, a FNC obtida é, por sua vez, mapeada para o formato DIMACS pelo bloco Mapeador para Formato DIMACS. Uma estrutura do tipo Hash é utilizada por tal submódulo para mapear os números DIMACS aos seus literais correspondentes. Para cada linha da tabela, o valor da sua chave de acesso corresponde ao inteiro DIMACS que representa o literal na FNC. Tomando como exemplo a fórmula em DIMACS apresentada na seção 3.1.2 com os literais $p_1 = 1$, $p_2 = 2$ e $p_3 = 3$, tem-se que a primeira linha da tabela hash tem como chave o inteiro 1 cujo conteúdo é o literal p_1. O mesmo ocorre para a segunda e terceira linhas, cujas chaves serão, respectivamente, o inteiro 2 e 3 que apontarão para os literais p_2 e p_3.

Na sequência, verifica-se se foi obtida uma FNC no formato DIMACS codificada para $t = N$ (i.e, a fórmula lógica proposicional ψ calculada para o *makespan* N). Em caso positivo, o módulo Solucionador SAT é invocado com a fórmula lógica ψ. Em caso negativo, a execução retorna ao submódulo Calculador de Axiomas que irá calcular novos axiomas para o instante de tempo t+1. Ações e símbolos proposicionais são criados a cada nova iteração tendo os seus respectivos atributos de tempo configurados para serem iguais ao instante de tempo corrente. Os novos axiomas a serem calculados são obtidos considerando apenas as ações e símbolos proposicionais criados para o tempo atual. Ao final de cada iteração, a FNC no formato DIMACS obtida no tempo t+1 é conectada à FNC obtida no tempo t.

5.3 MÓDULO SOLUCIONADOR SAT

O módulo Solucionador SAT é responsável por buscar por todos os modelos σ que satisfazem a fórmula lógica proposicional ψ recebida. Este módulo, como ilustra a figura 10, foi implementado através de dois submódulos. O submódulo *Invocador* faz uso de métodos da classe *Runtime* do Java (os quais permitem invocar e executar processos externos à aplicação Java) para que qualquer solucionador SAT completo da literatura pudesse ser invocado pelo protótipo. O *Invocador* recebe a fórmula lógica ψ no formato DIMACS e a escreve em um arquivo de saída txt que, por sua vez, é enviado ao solucionador SAT *bdd_minisat_all* (TODA; SOH, 2015).

Este solucionador mapeia uma FNC no formato DIMACS a um diagrama de decisão binária correspondente e o usa como uma estrutura para guiar a busca por todos os modelos σ de modo eficiente. Heurísti-

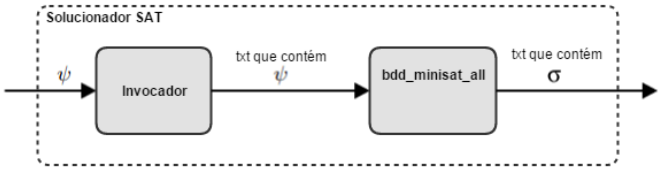

Figura 10 – Submódulos do Solucionador SAT.

cas para podar o espaço de busca baseadas em aprendizado de cláusulas e análise de conflito entre literais são empregadas a fim de que a memória ocupada pela estrutura seja a mais compacta possível permitindo que todos os modelos sejam encontrados o mais rápido possível.

Os modelos σ encontrados são impressos em um arquivo de saída txt e são dados na forma de sequências crescentes de números inteiros os quais refletem o conjunto de números DIMACS obtidos na etapa anterior. Se o solucionador SAT interpreta um certo literal como semanticamente verdadeiro, então o número DIMACS a ele associado é descrito no modelo como positivo, ao passo que se um literal é interpretado semanticamente como falso, então o número DIMACS a ele associado é descrito no modelo como negativo.

Se o solucionador SAT não encontrar nenhum modelo, então o arquivo de saída é deixado vazio (para este caso, alguns solucionadores escrevem a palavra *unsatisfiable* no arquivo de saída), então o protótipo interrompe a sua execução informando ao usuário que o problema é insatisfazível em relação ao *makespan* N. Se o arquivo contiver pelo menos um modelo, então o problema é considerado satisfazível e o fluxo de execução passa para o módulo Decodificador de FNC.

5.4 MÓDULO EXTRATOR DE PLANOS

O módulo *Extrator de Planos* é responsável por traduzir o conjunto de modelos que recebe em um conjunto de planos determinísticos. Entretanto, em testes preliminares verificou-se que parte dos modelos encontrados não poderiam ser interpretados como planos, pois para alguns instantes de tempo tais modelos: (*i*) não apresentavam ações a serem executadas; ou (*ii*) determinavam a execução de qualquer ação que tenha seus pré-requisitos satisfeitos. Sob a ótica do planejamento clássico, as ações de um plano são executadas imediatamente uma após

a outra para todo instante de tempo, o que invalida o caso *i*. No caso *ii*, devido a um excesso de tempo, ações a mais podem ser executadas, inclusive aquelas que não contribuem para o alcance dos objetivos, o que implica em excesso de esforço para obter G, desperdício de tempo, replicação e geração de efeitos desnecessários.

Verificou-se que quatro fatores estavam ligados a tais problemas: o axioma de exclusão mútua (AEM - ver definição 3.12), o tamanho do *makespan* N, o número necessário de ações para formar planos válidos e a ausência de efeitos negados. O axioma de exclusão mútua permite que, no máximo, uma ação seja escolhida a cada instante de tempo. Se o tamanho do *makespan* for maior do que o necessário para que um certo plano seja encontrado, então, como consequência do AEM, os instantes de tempo restantes poderão ou não serem preenchidos pelo solucionador SAT com quaisquer ações que tenham seus pré-requisitos satisfeitos ou simplesmente deixados vazios. Como o solucionador SAT usado neste trabalho é do tipo completo, então todos os modelos possíveis serão auferidos, incluindo aqueles que apresentam as características *i* e *ii*.

Considere que o *makespan* escolhido para o exemplo 7 da seção 4.4 seja N = 4, logo há um instante de tempo que excede os três instantes de tempo necessários para que os planos que solucionam aquele problema sejam encontrados. A tabela 1 mostra planos que podem ser obtidos através de modelos onde instantes de tempo não foram ocupados, isto é, instantes de tempo negligenciados (modelos com a característica *i*), ou seja, para tais instantes de tempo o solucionador SAT não escolheu ação alguma para ser executada. O símbolo − representa os instantes de tempo negligenciados.

t_0	t_1	t_2	t_3
LVDop	RVDop	PVCop	−
LVDop	RVDop	−	PVCop
−	LVDop	RVDop	PVCop

Tabela 1 – Exemplo 1 de modelos que devem ser evitados.

Note que os planos da tabela 1 são idênticos, pois as sequências de ações a serem executadas são as mesmas diferenciando-se, apenas, em relação aos instantes de tempo para os quais tais ações foram escolhidas para executarem. As ações de um plano serão salvas em uma fila sem o suporte a instantes de tempo negligenciados. Deste modo, o Decodificador de FNC obteria múltiplos planos idênticos, caso salvasse os planos da tabela 1, o que implicaria em uma quantidade maior de memória a ser utilizada para armazenar desnecessariamente tais pla-

nos, pois basta que um deles seja inserido na árvore binária para que as demais se tornem nulos ao serem escolhidas em uma nova inserção (ver algoritmo 2).

A tabela 2 mostra outros exemplos de planos obtidos para o *makespan* N = 4, onde o instante de tempo excedido foi preenchido com a ação $\texttt{PVCop}_2$.

t_0	t_1	t_2	t_3
LVDop	RVDop	PVCop	LVDop2
LVDop	RVDop	LVDop2	PVCop
LVDop	LVDop2	RVDop	PVCop
LVDop2	LVDop	RVDop	PVCop

Tabela 2 – Exemplo 2 de modelos que devem ser evitados.

Note que LVDop2 pôde ser inserido em qualquer posição dos planos justamente porque seus pré-requisitos (que são os mesmos de LVDop) são válidos para todos os instantes de tempo. Nestas condições, para cada instante, pelo menos, duas ações puderam ser executadas. É possível perceber que, quanto mais ações puderem ser executadas em um dado instante de tempo e quanto mais instantes de tempo sobrarem para a obtenção de um certo plano, então tal plano derivará outros planos (idênticos ou não) intercalando a ele novas ações ou negligenciando instantes de tempo.

Os planos da tabela 1 poderiam ser evitados se o axioma de exclusão mútua fosse estendido de modo a garantir que, para cada instante de tempo, uma e somente uma ação fosse escolhida para executar. O primeiro plano da tabela 2 poderia ser evitado caso o solucionador SAT utilizado[1] implementasse um mecanismo de subsunção de modelos, isto é, quando apenas modelos parciais são retornados, onde um modelo parcial refere-se a uma parte de um modelo que é suficiente para satisfazer uma fórmula lógica. Para o referido plano, um mecanismo de subsunção retornaria apenas um modelo parcial que originaria a sequência (LVDop, RVDop, PVCop). Um modelo parcial pode ser o sufixo de vários modelos, deste modo, os modelos que apresentam o mesmo sufixo são subsumidos e apenas uma cópia do sufixo é retornada. Infelizmente, para este trabalho não se encontrou uma forma de evitar planos como os últimos três da tabela 2 (i.e, planos nos quais foram inseridas ações quaisquer preenchendo instantes de tempo em excesso).

[1] Todos os solucionadores SAT completos testados neste trabalho não realizavam subsunção de modelos.

Como a função f' não está adaptada para suportar o axioma de exclusão mútua com as características anteriormente comentadas e o solucionador SAT completo utilizado não implementada um mecanismo de subsun ção de modelos, então decidiu-se implementar no módulo Decodificador de FNC um mecanismo para subsumir modelos e eliminar planos com instantes de tempo sem ações a serem executadas. Tal mecanismo funciona da seguinte forma: primeiramente, lê-se, em sequência, os inteiros DIMACS de um dado modelo ignorando os números assinados como negativos (i.e, semanticamente falsos). Quando um inteiro positivo é lido verifica-se se o literal que ele representa é uma referência para uma ação (já que literais também podem representar fatos que compõem os estados do mundo) determinando se a tabela de ações possui uma chave cujo valor é o próprio inteiro DIMACS. Em caso positivo, a ação é recuperada, adicionada em uma fila (estrutura que representa um plano) e a ordem de leitura daquele literal é salva. Convém dizer que a ordem de leitura é considerada apenas para os literais que representam ações e que o seu valor se inicia em zero, logo, nessas circunstâncias, a ordem do primeiro literal lido é zero, a do segundo é um e assim por diante.

O instante de tempo para o qual a ação recuperada foi criada é comparado com a ordem de leitura corrente. Se ambos forem iguais, então a ação foi escolhida para ser executada no instante de tempo imediatamente após o tempo anterior. Caso contrário, pelo menos, um instante de tempo foi negligenciado. Analisando o segundo plano da tabela 2, a ordem de leitura para `LVDop` é zero e para `RVDop` é um. Entretanto, `RVDop` foi escolhido para executar no tempo 2, logo esse plano será descartado.

Em seguida, verifica-se se a ação corrente alcança o conjunto de objetivos G. Para isso, um conjunto X para acumular os efeitos de cada ação é criado. Deste modo, $Pre(a)$ é adicionado ao conjunto X para, então, determinar se $G \subseteq X$. Em caso positivo, e se houverem mais ações a serem executadas em tempos posteriores, então o plano é descartado pois ele não é fruto de um modelo parcial. Este processo se repete até que todos os literais do modelo sejam lidos, então o plano obtido é salvo em uma lista. Ao final da análise de todos os modelos, a lista de planos é enviada ao módulo Fusor de Planos.

5.5 MÓDULO FUSOR DE PLANOS

O Módulo Fusor de Planos implementa o algoritmo 2 descrito na seção 4.2.2 para construir a árvore de decisão binária. Como parte da construção da árvore, o algoritmo *Merge Sort* também foi implementado para ordenar os planos segundo os critérios de similaridade, tamanho e custo. Merge sort é, inicialmente, chamado para ordenar a lista de planos pelo tamanho. Em seguida, verifica-se quantos planos possuem o mesmo tamanho daquele que ocupa a posição inicial da lista (isto é, o menor tamanho possível que um plano pode assumir naquela lista). Se apenas um plano possui o menor tamanho, então o primeiro plano da lista é retornado e a construção da árvore é iniciada a partir dele. Caso contrário, o subconjunto formado pelos planos que possuem o menor tamanho é ordenado segundo o custo de forma crescente. Finalmente, o primeiro plano do subconjunto é retornado garantindo que o primeiro plano a ser executado pelo agente tenha como característica o menor tamanho e custo possíveis.

Quando um plano é escolhido para ser inserido na árvore, chamadas recursivas são realizadas para adicionar uma a uma as suas ações à árvore. Quando as ações são adicionadas nodos de acerto e falha são criados. A um nodo de falha adiciona-se, caso exista, o melhor plano para ser alternativamente tentado pelo agente caso tal nodo seja alcançado em tempo de execução. Para isso, o método *getPlanoAlternativo*, um método de programação dinâmica, foi implementado como uma solução adaptada do *problema da maior subsequência comum* (*longest commom subsequence problem* - LCS) (ZHU et al., 2016).

Neste problema, deseja-se encontrar a maior subsequência comum, não necessariamente contígua, a duas sequências X e Y. O método getPlanoAlternativo testa cada plano (sequência X) da lista de planos com a sequência de ações que parte da raiz da árvore binária e alcança o nodo de falha corrente (sequência Y) a fim de determinar o grau de similaridade (tamanho da sequência comum) entre ambas as sequências. Em seguida, os planos são ordenados segundo o grau de similaridade e sobre tais planos obtém-se um novo subconjunto formado apenas pelos planos com o maior grau de similaridade que um plano da lista original pode assumir. A posição na lista do último plano inserido no subconjunto prévio é salva para que, caso haja necessidade, um novo subconjunto formado pelos planos que possuem o próximo maior grau de similaridade seja obtido e submetido à análise de tamanho e custo.

Por sua vez, o subconjunto prévio passa a ser ordenado pelo tamanho e os planos que nele possuem o menor tamanho possível pas-

sam a formar um novo subconjunto que é, então, ordenado pelo custo. Assim como na análise da similaridade, a posição do último plano adicionado no último subconjunto obtido é salva para que, caso haja necessidade, um novo subconjunto formado pelos planos que possuem o próximo menor tamanho seja obtido e submetido a análise de custo. Então, o subconjunto ordenado pelo custo é percorrido a partir do seu início a fim de determinar se algum plano respeita o critério de seleção *iv* (não possuir ações de acerto que são contraparte de ações de falha em Y). Se tal plano existe, então ele é selecionado e o processo de construção da árvore prossegue. Caso contrário, a análise para encontrar o melhor plano alternativo retorna para o subconjunto anterior a fim de obter um novo subconjunto constituído pelos planos com o próximo menor tamanho possível.

Se naquele subconjunto não houverem mais planos, novamente, o método volta a sua análise para o conjunto ordenado pela similaridade a fim de obter um novo subconjunto formado pelos planos com a próxima maior similaridade. Então, a análise avança novamente para as análises de tamanho e custo. Este processo se repete até que algum plano alternativo tenha sido encontrado ou quando todos os planos da lista original tenham sido analisados e verificou-se que nenhum deles pode ser considerado como plano alternativo. Por sua vez, a construção da árvore finaliza quando não existem mais planos na lista de planos para serem inseridos na árvore.

5.6 TESTES E RESULTADOS PRELIMINARES

O protótipo implementado foi avaliado empiricamente através de três canários a fim de determinar a sua escalabilidade. Cada cenário é composto por dez configurações, onde cada configuração é, por sua vez, composta pelo número de ações resultante do mapeamento de W' para A (*ações*), número obtido de planos ($|\rho'$s$|$), tempo de mapeamento de planejamento para SAT (*map. SAT*), tempo de busca pelos modelos (*busc.* σ's), tempo de impressão dos modelos (*imp.* σ's), tempo de tradução dos modelos encontrados para planos (*map.* σ's *p/* ρ's), tempo de construção da árvore binária (*cons. árv*) e tempo total de execução do protótipo (*t. total* - soma de todos os tempos obtidos) conforme apresentados nas tabelas 3, 4 e 5, respectivamente, onde a primeira linha corresponde à primeira configuração, a segunda linha corresponde à segunda configuração, e assim por diante. Os tempos foram medidos em segundos (s), minutos (m) e horas (h). Ademais, nas tabelas, o

símbolo − é utilizado para indicar que um determinado resultado não foi possível de ser obtido em menos de 24 horas.

Para cada configuração, variou-se de cem em cem o número de operações disponíveis em W', de modo que, a configuração inicial e final para cada cenário foi, nesta ordem, composta por cem e mil operações. As operações em W', que quando combinadas em alguma sequência de invocação formarão um plano esperado, serão chamadas de operações úteis e estão descritas a seguir para cada cenário. As demais operações em W' serão chamadas de operações semi-úteis, pois são retornadas pelo motor de busca por combinarem parcialmente/totalmente com os parâmetros de entrada/saída de outras operações em W' ou com a requisição do usuário. Tais operações quando combinadas com outras em uma dada sequência de invocação não formarão os planos esperados.

As operações úteis foram replicadas progressivamente para cada nova configuração de acordo com uma razão que, por sua vez, variou para cada cenário. As cópias funcionam como operações alternativas e podem substituir umas às outras, formando novos planos. Se um certo plano possui tamanho n e se cada uma das operações que o compõem possui k réplicas, então serão derivados k^n planos. Deste modo, a cada nova configuração, o número de planos foi maior em relação à configuração anterior. Cada configuração foi testada 10 vezes, contabilizando 300 testes para os três cenários. Os testes foram realizados em um computador com quatro gigabytes de memória RAM, processador Intel Core i5 5200U 2.2 GHz, sistema operacional Ubuntu 16.10 de 64 bits. A seguir, os testes são apresentados em maiores detalhes.

5.6.1 Cenário 1

Os testes realizados neste cenário consistiram em obter todos os planos para o problema apresentado no exemplo 7 da seção 4.4 no qual um usuário requisitou um serviço para comprar passagens aéreas para viajar entre duas cidades distintas. Destaca-se que as operações `ListarVoosDisponíveis (LVDop)`, `ReservarVoosDisponíveis (RVDop)` e `PagamentoViaCrédito (PVCop)` foram replicadas progressivamente para cada nova configuração de acordo com uma razão igual a 3. Para a primeira configuração, W' contabilizou 3 cópias de `LVDop`, `RVDop` e `PVCop`, isto é, 9 operações úteis. A tabela 3 sumariza os resultados obtidos para o cenário 1.

A tabela 3 mostra que foram obtidos planos no intervalo de 27 a 27 mil planos e o tempo total de execução variou de 0,2862 segundos

ações	$\|\rho's\|$	map. SAT	busc. $\sigma's$	imp. $\sigma's$	$\sigma's$ p/ $\rho's$	cons. árv	t. total	txt
100	27	0,0837s	0,0005s	0,0004s	0,0279s	0,1737s	0,2862s	6.4 KB
200	216	0,1065s	0,0005s	0,0033s	0,1339s	0,2785s	0,5227s	90,5 KB
300	729	0,1869s	0,0011s	0,0118s	0,1640s	0,7989s	1,1627s	349,9 kB
400	1.728	0,2087s	0,0022s	0,0237s	0,2444s	2,8622s	3,3412s	1,1 MB
500	3.375	0,2307s	0,0031s	0,0537s	0,3652s	7,9743s	8,6270s	2,5 MB
600	5.832	0,2716s	0,0041s	0,1062s	0,4791s	22,7620s	23,6230s	5,2 MB
700	9.261	0,3129s	0,0058s	0,1921s	0,5334s	1,1224m	1,1398m	9,4 MB
800	13.824	0,3725s	0,0073s	0,3051s	1,2887s	2,8194m	2,8522m	16,0 MB
900	19.683	0,4325s	0,0082s	0,4686s	1,7593s	6,2581m	6,3024m	25,4 MB
1000	27.000	0,5212s	0,0099s	0,7628s	2,1491s	13,9120m	13,9693m	38,5 MB

Tabela 3 – Resultados para o cenário 1.

a 13,9693 minutos. 60% dos testes tiveram tempo total de execução abaixo de 1 minuto e a partir da quinta configuração o tempo de construção da árvore passou a consumir mais 90% do tempo total.

A figura 11 ilustra, no formato de barras, o crescimento do tempo de construção das árvores binárias (barras vermelhas) e crescimento do tempo de execução total do protótipo (barras azuis). Cada valor no eixo das abscissas corresponde a uma configuração da tabela 3 e, para cada barra de uma dada configuração, o seu respectivo valor no eixo das ordenadas corresponde ao seu tempo de obtenção.

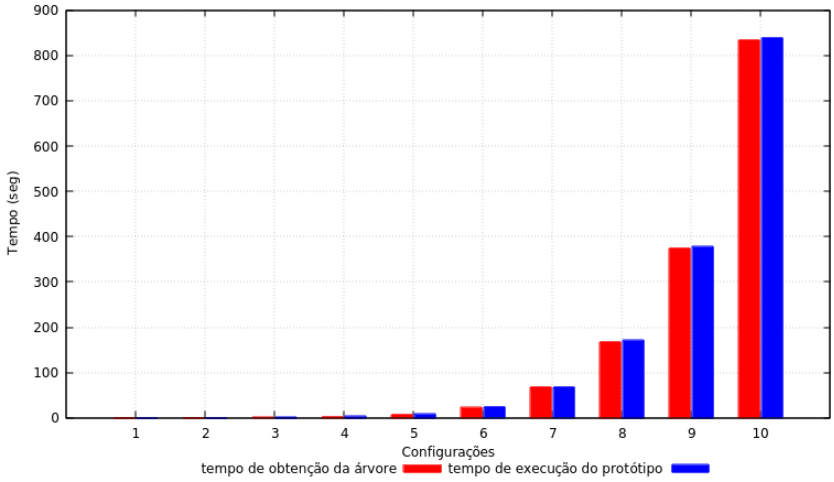

Figura 11 – Tempos para a construção de árvores binárias e execução total do protótipo no cenário 1.

A figura 11 mostra que, para qualquer valor do eixo das abscissas,

a diferença entre o tempo total de execução do protótipo e o tempo de obtenção da árvore é quase imperceptível. É possível notar que o tempo de construção da árvore binária para os últimos dois pontos cresce muito mais rápido do que para os demais pontos. O esforço computacional para a obtenção da árvore binária ocupa, praticamente, todo o tempo de execução do protótipo. Isso se dá, principalmente, pela necessidade de ordenação dos planos de acordo com a compatibilidade com cada nodo de falha gerado, tamanho e custo.

5.6.2 Cenário 2

A fim de que o desempenho do protótipo pudesse ser avaliado na presença de planos de diferentes tamanhos, o cenário um foi estendido através da adição de duas novas ações. Deste modo, foi possível obter planos de comprimento um, dois e três.

As novas ações adicionadas foram as seguintes:

`ComprarPassagemAérea (CPAop)`: busca, reserva e compra dois voos, um de ida e outro de volta.
$I = \{$`dataPartida,dataRetono,aeroportoOrigem,aeroportoDestino,classe, preçoMáximo,cartaoDeCrédito`$\}$;
$O = \{$`transação`$\}$;
$ND = \langle\{$`transação.estado` $=$ `aprovado` $\wedge$
`cartaoDeCrédito.saldo` $=$ `cartaoDeCrédito.saldo`$-$`valor`$\}$,
$\{$`transação.estado` $\neq$ `aprovado` $\vee$
`cartaoDeCrédito.saldo` $\neq$ `cartaoDeCrédito.saldo`$-$`valor`$\}\rangle$;
$\mathcal{K} = 3$.

`ObterPassagemAérea (OPAop)`: busca e reserva dois voos, um de ida e outro de volta, mas não efetua o pagamento.
$I = \{$`dataPartida,dataRetono,aeroportoOrigem,aeroportoDestino,classe, preçoMáximo`$\}$;
$O = \{$`voosReservados,preço`$\}$;
$Pre = \{\}$;
$ND = \langle\{$`voosDisponíveis.tamanho`$> 0\}$, $\{$`voosDisponíveis.tamanho`$\leq 0\}\rangle$;
$\mathcal{K} = 2$.

Os planos esperados com a adição destas duas novas ações são os seguintes:

- $\rho_1 = ($`CPAop`$)$

- ρ_2 = (OPAop,PVCop)
- ρ_3 = (LVDop,RVDop,PVCop).

Entretanto, como visto na seção 5.4, quando o *makespan* for maior do que o necessário para que um certo plano seja encontrado, então, para alguns instantes de tempo, alguns modelos retornados por um solucionador SAT completo que não realize subsunção de modelos não apresentam ações a serem executadas ou determinam a execução de qualquer ação que tenha seus pré-requisitos satisfeitos. Neste segundo caso, a ação pode não contribuir com o alcance dos objetivos (o que equivale a realizar uma tarefa desnecessária para o alcance dos objetivos) ou gerar efeitos obtidos em instantes anteriores (o que equivale a reexecutar uma tarefa desnecessariamente).

Nessas circunstâncias, os seguintes planos são exemplos de planos que também serão obtidos e que devem ser subsumidos:

- Em ρ_4 = (CPAop,LVDop,RVDop) as ações LVDop e RVDop são executadas desnecessariamente já que os objetivos são alcançados no primeiro instante de tempo.
- Em ρ_5 = (−,−,CPAop) os dois primeiros instantes de tempo foram negligenciados e o alcance dos objetivos foi atrasado para o terceiro instante de tempo.
- Em ρ_6 = (OPAop,OPAop,PVCop) a ação OPAop é executada uma segunda vez desnecessariamente.
- Em ρ_7 = (OPAop,−,PVCop) o segundo instante de tempo foi negligenciado e, assim como ρ_5, o alcance dos objetivos foi atrasado para o terceiro instante de tempo.

Deste modo, para este cenário, houve a necessidade de que um mecanismo de subsunção fosse aplicado. A tabela 4 sumariza os resultados obtidos para o cenário 2, onde a coluna $\sigma's\ p/\ \rho's$ considera o tempo para subsumir e traduzir os modelos encontrados pelo solucionador SAT em planos, e a coluna $|\rho'\text{s}|$ refere-se ao número de planos que não foram filtrados pelo mecanismo de subsunção e, portanto, foram utilizados para a construção de árvores binárias.

Neste cenário, para a primeira configuração, W' contabilizou 3 cópias de LVDop, RVDop, PVCop, CPAop e OPAop, totalizando 15 operações úteis. As operações foram replicadas progressivamente para cada nova configuração de acordo com uma razão igual a 3. A tabela 4 mostra

ações	$\|\rho's\|$	map. SAT	busc. $\sigma's$	imp. $\sigma's$	$\sigma's$ p/ $\rho's$	cons. árv.	t. total	txt
100	273	0,3510s	0,0098s	0,0999s	1,2073s	0,2171s	1,8851s	5,2 MB
200	2058	0,9115s	0,0204s	1,7682s	4,4924s	0,9290s	8,1211s	79,8 MB
300	6813	1,6475s	0,0572s	8,8616s	34,932s	39,9189s	1,4236m	397,0 MB
400	15996	2,0214s	0,0976s	28,3881s	1,5873m	8,3323m	10,4280m	1,3 GB
500	31065	2,5142s	0,1284s	58,6932s	3,2012m	39,6471m	43,8705m	3,2 GB
600	53478	3,1004s	0,1584s	1,7122m	8,9616m	2,0582h	2,2370h	8,0 GB
700	84693	3,6678s	0,1992s	3,3041m	13,1365m	6,5371h	6,8121h	12,8 GB
800	126168	4,8835s	0,2588s	5,6274m	19,3551m	12,6667h	13,0845h	22,0 GB
900	179361	7,6464s	0,3784s	9,1597m	32,6342m	–	–	35,5 GB
1000	245730	11,9212s	0,5657s	14,255m	1,2208h	–	–	54,4 GB

Tabela 4 – Resultados para o cenário 2.

que foram obtidos planos no intervalo de 273 a pouco mais de 245 mil planos. Entretanto, não foi possível obter o tempo total de execução do protótipo para a nona e décima configurações, pois o tempo de obtenção da árvore binária ultrapassou 24 horas de execução. Neste cenário apenas 20% das configurações ficaram abaixo de 1 minuto e a partir da quinta configuração o tempo de obtenção da árvore binária passou a ocupar 90% do tempo total de execução do protótipo. Entretanto, em relação ao cenário um, os tempos do solucionador SAT para imprimir os modelos encontrados foram maiores, ultrapassando os 5 minutos a partir da oitava configuração e chegando a 14 minutos na décima configuração.

O solucionador SAT gerou arquivos de saída que ocuparam mais 10 GB em disco. Em compensação, o tamanho máximo que um arquivo de saída obteve no cenário um foi de 38,5 MB. Isto ocorreu, principalmente, porque a falta de um mecanismo de subsunção permitiu que a explosão combinatória de modelos fosse impressa por completo nos arquivos de saída.

A figura 12 ilustra, no formato de barras, o crescimento do tempo de execução total do protótipo no cenário 1 (barras vermelhas) e do protótipo no cenário 2 (barras azuis).

Para cada barra, o seu valor no domínio corresponde à configuração de sua respectiva tabela (tabelas 3 e 4 para barras vermelhas e azuis, respectivamente), e o seu respectivo valor no eixo das ordenadas corresponde ao seu tempo de obtenção. Acima de 800 ações (8ª configuração) não se conhece o tempo de execução total do protótipo para o cenário 2, logo barras azuis não existem para as 9ª e 10ª configurações. O gráfico 12 evidencia que o tempo total para o protótipo no cenário 2 é consideravelmente superior ao tempo total do protótipo no cenário 1. Isso ocorre, principalmente, por dois motivos: (i) conforme o tamanho de W' aumenta, o número de planos a serem inseridos na árvore binária

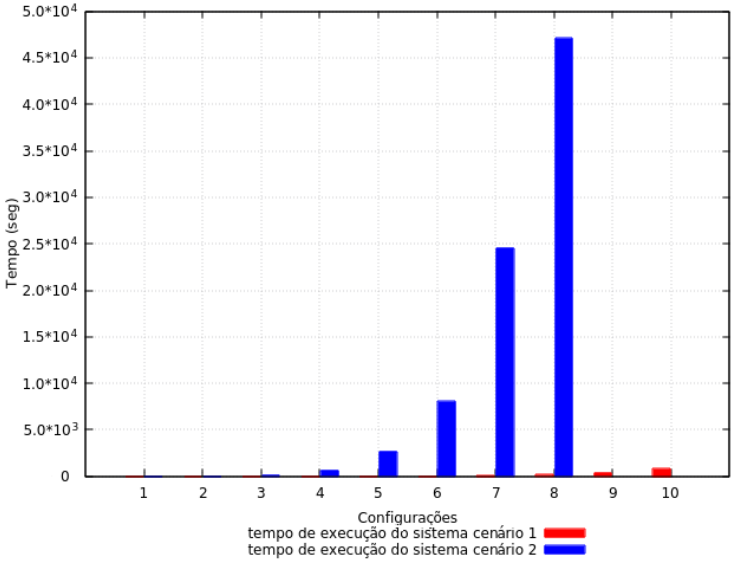

Figura 12 – Tempos de execução total do protótipo nos cenários 1 e 2.

no cenário 2 é muito maior do que o número de planos no cenário 1; e (*ii*) o tempo de ordenação dos planos no cenário 1 é relativamente menor do que o tempo de ordenação dos planos para no cenário 2. No caso do cenário 1, todos os planos têm o tamanho 3, o que faz com que a complexidade do tempo do algoritmo Merge Sort seja O(n) (melhor caso). No caso do cenário 2, os planos possuem tamanhos diferentes (de 1 a 3) o que faz com que a complexidade do algoritmo Merge Sort seja O(n log n) (caso médio e pior caso).

A figura 13 ilustra o crescimento do tempo de construção das árvores binárias (barras vermelhas) e do tempo total de execução do protótipo (barras azuis).

Os valores no eixo das ordenadas correspondem aos cenários da tabela 4 e, para cada barra de uma dada configuração, o seu respectivo valor no eixo das ordenadas corresponde ao seu tempo de obtenção.

Assim como no cenário 1, para qualquer valor do eixo das abscissas, a diferença entre o tempo total de execução do protótipo e o tempo de construção da árvore é quase imperceptível. Entretanto, o tempo de construção das árvores binárias no cenário 2 cresce muito mais rápido do que o tempo de construção das árvores binárias no ce-

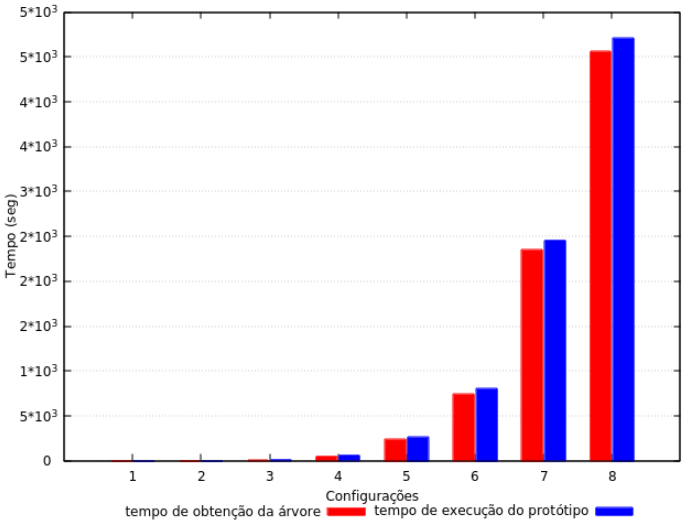

Figura 13 – Tempos para a construção de árvores binárias e execução total do protótipo no cenário 2.

nário 1, pois, para cada configuração, o número de planos obtidos no cenário 2 é muito maior do que o número de planos obtidos no cenário 1. Ademais, a variação de tamanhos dos planos obtidos no cenário 2 implica em um esforço computacional maior para ordená-los.

5.6.3 Cenário 3

Neste cenário, uma empresa busca criar um serviço que permita a usuários agendarem consultas médicas on-line. Para isso, o paciente deve descrever os sintomas que sente e o serviço indica a provável doença do usuário, um médico com a especialidade que possa confirmar o diagnóstico prévio e determinar um tratamento. O médico deve atender pelo plano de saúde do paciente, ter horário disponível na data desejada e atender na clinica médica mais perto da residência do paciente. As operações que formam W' são as seguintes (onde `= null` significa que não foi possível encontrar o valor para um determinado parâmetro):

`ObterProntuário (OPop):` retorna uma versão digital do prontuário do paciente.

$I = \{\texttt{login}\};$
$O = \{\texttt{prontuário,dadosPaciente}\};$
$Pre = \{\texttt{login} \neq \texttt{null}\};$
$ND = \langle\{\texttt{prontuário} \neq \texttt{null}\}, \{\texttt{prontuário} = \texttt{null}\}\rangle;$
$\mathcal{K} = 1.$

`DeterminarDoença (DDop):` determina a provável doença do paciente tomando como base uma lista de sintomas indicados pelo paciente, seu prontuário e dados pessoais como sexo e idade, por exemplo.
$I = \{\texttt{listaSintomas,prontuário,dadosPaciente}\};$
$O = \{\texttt{doença}\};$
$Pre = \{\texttt{listaSintomas>5}\};$
$ND = \langle\{\texttt{doença} \neq \texttt{null}\},$
$\{\texttt{doença} = \texttt{null}\}\rangle;$
$\mathcal{K} = 3.$

`DeterminarEspecialidade (DEop):` determina uma especialidade médica para tratar uma provável doença.
$I = \{\texttt{doença}\};$
$O = \{\texttt{especialidade}\};$
$Pre = \{\ \};$
$ND = \langle\{\texttt{especialidade} \neq \texttt{null}\},$
$\{\texttt{especialidade} = \texttt{null}\}\rangle;$
$\mathcal{K} = 3.$

`ObterMédicoComEspecialidade (OMEop):` determina o médico que possui a especialidade requerida, atenda pelo convênio do paciente e na clínica mais perto do seu endereço, tenha horário disponível na data desejada.
$I = \{\texttt{especialidade,convênio,dataConsulta,endereçoPaciente}\};$
$O = \{\texttt{médico,localAtendimento}\};$
$Pre = \{\texttt{especialidade} \neq \texttt{null} \wedge \texttt{convênio} \neq \texttt{null}\};$
$ND = \langle\{\ \neq \texttt{null}\},$
$\{\ \}\rangle;$
$\mathcal{K} = 2.$

`AgendarConsulta (ACop):` agenda uma consulta para o paciente.
$I = \{\texttt{dataConsulta,médico,localAtendimento,dadosPaciente}\};$
$O = \{\texttt{agendamento,valor}\};$
$Pre = \{\ \};$
$ND = \langle\{\texttt{doença} \neq \texttt{null}\},$
$\{\texttt{doença} = \texttt{null}\}\rangle;$

$\mathcal{K} = 2$.

`PagamentoViaCrédito (PVCop)`: realiza pagamento via crédito:
$I = \{$`cartaoDeCrédito, valor`$\}$;
$O = \{$`transação`$\}$;
$Pre = \{$`cartaoDeCrédito.saldo` $\geq$ `valor`$\}$;
$ND = \langle\{$`transação.estado` $=$ `aprovado` $\wedge$
`cartaoDeCrédito.saldo` $=$ `cartaoDeCrédito.saldo`$-$`valor`$\}$,
$\{$`transação.estado` $\neq$ `aprovado` $\vee$
`cartaoDeCrédito.saldo` $\neq$ `cartaoDeCrédito.saldo`$-$`valor`$\}\rangle$;
$\mathcal{K} = 1$.

Neste cenário, deseja-se testar o desempenho do protótipo para obter árvores binárias com planos de tamanhos maiores (comprimento seis) em relação aqueles obtidos nos cenários anteriores. Os planos esperados constituem-se da seguinte sequência de ações: (`OPop`,`DDop`,`DEop`,`OMEop`, `ACop`,`PVCop`).

A tabela 5 sumariza os resultados obtidos para o cenário 3. Para a primeira configuração, W' contabilizou apenas uma cópia de cada ação anteriormente apresentada, totalizando 6 ações úteis. Para cada nova configuração, o número de cópias de uma dada ação cresceu em uma razão igual a 1.

ações	$\lvert\rho'$s$\rvert$	map. SAT	busc. σ's	imp. σ's	σ's p/ ρ's	cons. árv.	t. total	txt
100	1	0,1145s	0,0004s	0,0006s	0,0148s	0,0093s	0,1396s	431 bytes
200	64	0,1721s	0,0010s	0,0020s	0,0662s	0,1582s	0,3995s	39,1 KB
300	729	0,2475s	0,0012s	0,0122s	0,1602s	1,5171s	1,9382s	576,6 KB
400	4096	0,3323s	0,0038s	0,0867s	0,3897s	20,283s	21,0955s	4 MB
500	15625	0,4242s	0,0109s	0,3398s	1,2969s	5,1675m	5,20203m	18 MB
600	46656	0,5411s	0,0062s	1,1663s	3,8359s	45,2267m	45,3191m	62,1 MB
700	117649	0,6730s	0,0254s	3,3471s	8,9370s	5,1181h	5,1217h	117,8 mb
800	262144	0,7935s	0,0523s	8,5425s	17,538s	–	–	443,3 mb
900	531441	0,9464s	0,0495s	18,511s	54,955s	–	–	994,3 MB
1000	1000000	1,8012s	0,0285s	39,352s	3,1352m	–	–	2,1 GB

Tabela 5 – Tempos para a construção de árvores binárias e execução total do protótipo no cenário 3.

A tabela 5 mostra que foram obtidos planos no intervalo de 1 a 1 milhão de planos. A grande quantidade de planos obtidos se deve ao fato de que uma dada ação pode ser substituída pelas suas réplicas, o que gera diferentes planos. Os tempos de execução do protótipo variaram de 0,1396 segundos a pouco mais de 5 horas. A partir da 8ª configuração não foi possível obter o tempo de construção da árvore binária e o tempo total. Nesta configuração, o número de planos ultrapassou os 262 mil, e fica evidente que com tal valor não é possível

construir uma árvore binária em menos de 24 horas já que, ao analisar o cenário dois, a partir de 179361 planos também não é possível obter uma árvore binária nas mesmas condições.

O tempo de impressão dos modelos e tamanho dos arquivos de saída do solucionador SAT para o cenário 3 foram menores em relação ao cenário 2 porque a quantidade de modelos encontrados no cenário 3 foi menor. Entretanto, no cenário 2 grande parte dos modelos encontrados foram subsumidos, fazendo com que o número total de planos usados para construir árvores binárias fosse menor em relação ao número total de planos para o mesmo fim no cenário 3 (onde cada modelo pôde ser interpretado como um plano). A explosão combinatória que gerou o número de planos no cenário 3 inviabilizou obter os testes por completo nos últimos 3 cenários. Ademais, 40% dos testes ficaram abaixo de 1 minuto e a partir da quarta configuração o tempo de obtenção da árvore binária toma mais de 90% do tempo total de execução do protótipo.

A figura 14 ilustra, no formato de barras, o crescimento do tempo de construção das árvores binárias (barras vermelhas) e crescimento do tempo de execução total do protótipo (barras azuis). Cada valor no eixo das abscissas corresponde a uma configuração da tabela 5 e, para cada barra de uma dada configuração, o seu respectivo valor no eixo das ordenadas corresponde ao seu tempo de obtenção.

Assim como nos demais cenários, para qualquer valor do eixo das abscissas, a diferença entre o tempo total de execução do protótipo e o tempo de obtenção da árvore é quase imperceptível. O tempo de construção das árvores binárias cresce muito mais do que o mesmo tempo para os demais cenários, pois o número de planos e os seus tamanhos obtidos no cenário 3 são maiores do que nos demais cenários o que implica em um maior esforço computacional para se obter árvores binárias.

Por fim, pode-se concluir que o protótipo implementado obteve bons resultados em testes preliminares mostrando que é possível de obter árvores binárias em tempos relativamente baixos e com uma quantidade grande e satisfatória de planos. Em torno de 1 minuto foi possível construir árvores binárias com até 9261, 6813 e 4096 planos nos cenários 1, 2 e 3, respectivamente, o que corresponde a uma média de 6723 planos. Em torno de 5 minutos foi possível construir árvores binárias com até 13824, 15996, 15625 planos nos cenários 1, 2 e 3, respectivamente, o que corresponde a uma média de 15148 planos. Entretanto, acima de 31000 planos o protótipo passou a tomar uma quantidade considerável de tempo para o seu processamento, chegando a quase 1

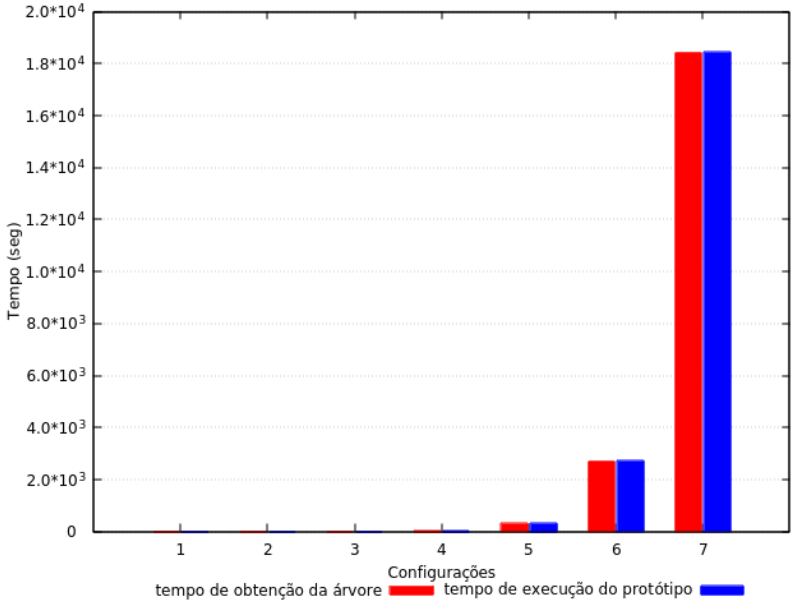

Figura 14 – Tempo total e tempo de construção da árvore binária para o cenário 3.

hora. Ademais, as maiores quantidades de planos obtidos foram 117649 e 126168 para os cenários dois e três, respectivamente, e acima destes valores não foi possível obter árvores binárias em menos de 24 horas.

5.7 COMPARATIVO ENTRE OS TRABALHOS

A tabela 6 sumariza qualitativamente as principais diferenças entre a proposta desta dissertação e os trabalhos relacionados apresentados no capítulo 2. As colunas indicam os trabalhos correlatos e as linhas indicam os critérios de avaliação, onde *sim* e *não* indicam se um trabalho atende ou não a um dado critério, respectivamente. Na tabela, K&F, M&R, H&Et, W&Et, T&H, A&P, K&C, K&R e ★ referem-se, nesta ordem, aos trabalhos de Kaholy e Fatatry (2016), Markou e Refanidis (2016), Huang et. al (2015), Wang, W. et. al (2015), Torrez e Holvoet (2014), Alferez e Pelechano (2013), Kuzu e Cicekli (2012), Klusch e Renner (2006) e a proposta desta dissertação. O símbolo – é usado quando não foi possível identificar a técnica utilizada para se ob-

ter um plano. Ademais, PBM, PP, PND, GR referem-se as técnicas de planejamento baseado em modelos, probabilístico, não-determinístico e grafo relaxado, respectivamente.

Critérios	K&f	H&Et	W&Et	M&R	T&H	A&P	K&C	K&R	★
Considera algum método para determinar compatibilidade semântica entre parâmetros	não	não	não	não	não	sim	sim	sim	sim
Considera alguma métrica (custo, tamanho, utilidade) para decidir qual plano executar	não	não	sim	sim	sim	não	não	não	sim
Obtém $k > 1$ planos alternativos antes da fase de execução	não	não	não	sim	não	não	não	não	sim
Trata falhas relacionadas a Web services indisponíveis	não	sim	sim	sim	sim	sim	sim	sim	sim
Trata falhas relacionadas à produção de efeitos indesejáveis	sim	sim	sim	sim	não	não	sim	não	sim
Necessita aplicar replanejamento durante a fase de execução	sim	sim	não	não	não	sim	sim	sim	não
Técnica de busca por plano	-	PBM	-	PP	-	-	GR	GR	SAT
Tipo de solução obtida	fraca	fraca	fraca	fraca	fraca	fraca	fraco	fraca	Forte cíclica

Tabela 6 – Comparativo entre os trabalhos relacionados.

Nota-se que a maioria dos trabalhos negligencia o fato de que retornar a um estado anterior da composição, executar uma operação alternativa, replanejar uma solução ou findar a execução da composição em insucesso necessita que os efeitos produzidos pelas operações que alteram o mundo real devem ser revertidos. Também é possível notar que parte dos trabalhos não considera compatibilidade semântica entre os parâmetros.

Os únicos trabalhos a obterem $k > 1$ planos alternativos antes da fase de execução são o presente trabalho e Markou e Refanidis (2016), logo ambos *não* necessitam aplicar replanejamento quando uma falha impede uma operação de produzir os efeitos desejados. Os trabalhos de Wang, W. et. al (2015) e Torrez e Holvoet (2014) também não aplicam replanejamento, entretanto esta estratégia poderia complementar o leque de estratégias implementadas por tais trabalhos e ser acionada quando as demais estratégias falharem ou não puderem ser aplicadas. Ademais, com exceção da proposta desta dissertação, os demais trabalhos obtêm, segundo a ótica do planejamento, soluções fracas, isto é, soluções que não garantem que os objetivos do usuário sejam alcançados. O trabalho de Markou e Rafanidis (2016) não provê nenhum mecanismo para que, ao alcançar um estado morto, o agente de planejamento execute um novo caminho na árvore que possibilite o alcance dos seus objetivos. Deste modo, a árvore de decisão binária adotada pelos autores acaba sendo, como um todo, um plano condicional fraco formado por planos fracos menores. Apenas a presente proposta obtém uma solução forte cíclica, isto é, garante que para todo estado do mundo (mais especificamente um estado do mundo, no contexto desta

dissertação, equivale a um estados da composição resiliente resultante) é possível alcançar um estado de objetivos (uma folha tracejada na árvore binária), entretanto paga-se o preço de que ciclos sejam executados até que um conjunto de efeitos desejados sejam obtidos (mas não há garantia de que tal conjunto seja efetivamente obtido).

No próximo capítulo serão apresentadas as conclusões e perspectivas de trabalhos futuros.

6 CONCLUSÃO

Este trabalho apresentou uma proposta para a obtenção de composições resilientes de Web services, isto é, composições que são capazes de contornar problemas de execução e fornecer seus serviços de modo contínuo frente à ocorrência de falhas em ambientes dinâmicos e não-determinísticos. A abordagem proposta combina planejamento não-determinístico e SAT para que k planos determinísticos alternativos (composições alternativas de Web services) que solucionam uma dada requisição de usuário sejam obtidos e fundidos em uma estratégia de contingência na forma de uma árvore de decisão binária (composição resiliente de Web services resultante).

A estratégia de contingência tem como finalidade permitir a motores de execução de processos acompanhar o progresso da execução da composição resiliente resultante e lidar com problemas de execução mediante a rápida seleção, com custo mínimo, de uma composição alternativa e compatível com aquela que não foi bem sucedida. Ademais, a árvore binária obtida neste trabalho é, segundo a ótica do planejamento não-determinístico, classificada como uma solução forte cíclica, ou seja, uma solução que garante que para todo estado do mundo é possível alcançar um estado de objetivos, entretanto paga-se o preço de que ciclos sejam executados até que um conjunto de efeitos desejados sejam obtidos (mas não há garantia de que tal conjunto seja efetivamente obtido).

As principais contribuições deste trabalho foram: 1) um método automático que 1.a) obtém k planos alternativos determinísticos para a elaboração de composições resilientes de Web services tirando vantagem da bem conhecida e robusta técnica de planejamento SAT; 1.b) evita a necessidade de replanejamento em tempo de execução, uma vez que todas os planos de comprimento máximo N são obtidos antecipadamente; 2) uma estratégia de contingência que 2.a) permite a motores executar diferentes composições alternativas de Web services mediante à ocorrência de problemas de execução; 2.b) evita que as composições fiquem presas em estados mortos; 2.c) aumenta as chances de que os objetivos do usuário sejam alcançados em ambientes dinâmicos e não-determinísticos.

Um protótipo que foi desenvolvido em Java implementa parcialmente a arquitetura do framework proposto no capítulo 4. Testes sobre a sua escalabilidade foram realizados e resultados preliminares foram obtidos. O sistema obteve bons resultados realizando todas as etapas

para a construção de árvores binárias em tempos relativamente baixos e com uma quantidade grande e satisfatória de planos. Em torno de um minuto foi possível construir árvores binárias com uma média de 6723 planos, e em torno de cinco minutos árvores binárias foram construídas com uma média de 15148 planos. Acima de tais medidas o sistema passou a tomar frações de horas para terminar o seu processamento, e para algumas configurações ultrapassou 24 horas de execução.

Entre as limitações deste trabalho pode-se destacar a falta de: 1) aplicação de métricas mais flexíveis (relaxadas) para determinar a compatibilidade semântica entre operações de Web services; 2) aplicação de diferentes estratégias para contornar problemas de execução (como tentar re-executar a operação que falhou ou obter uma sequência de operações que produzam os mesmos efeitos da operação que falhou quando não existe uma operação alternativa para executar em seu lugar, por exemplo); 3) execução de operações em paralelo.

Como trabalhos futuros pode-se destacar: 1) a extensão da função 4.3 de mapeamento de planejamento para SAT de modo a 1.a) permitir a invocação e execução de ações em paralelo; 1.b) garantir que ações só possam ser escolhidas para executar em um instante de tempo t, se e somente se, uma ou mais ações forem escolhidas para executar no instante de tempo t-1; 2) Estudar diferentes solucionadores SAT completos a fim de determinar aquele que melhor atenda as necessidades deste trabalho realizando subsunção de modelos; 3) Estudar o uso de simplificadores de FNCs a fim de se obter fórmulas lógicas proposicionais mais compactas; 4) Obter composições de Web services de acordo com diferentes critérios de compatibilidade entre operações semânticas;

5) Testar o sistema implementado com *benchmarks* da literatura; 6) Determinar um número limiar de planos que devem ser obtidos para a construção de árvores binárias; 7) Fazer uso de alguma função probabilística, como a utilidade esperada, como critério adicional para selecionar um plano a ser inserido a partir de um nodo de falha. 8) Implementar a fase de tradução da estratégia de contingência para um processo executável e testá-lo em ambientes dinâmicos e não-determinísticos para que se determine a eficiência da proposta desta dissertação em alcançar os objetivos do usuário frente à ocorrência de falhas.

REFERÊNCIAS

AHO, A.; SETHI, R.; LAM, S. **Compiladores: princípios, técnicas e ferramentas**. [S.l.]: LONGMAN DO BRASIL, 2008. ISBN 9788588639249.

ANGARITA, R.; RUKOZ, M.; MANOUVRIER, M. Dynamic composite web service execution by providing fault-tolerance and qos monitoring. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 8954, p. 371–377, 2015.

Apache Software Foundation. **Orchestration Director Engine**. 2013. Disponível em: <http://ode.apache.org>.

BARTALOS, P.; BIELIKOVA, M. Automatic dynamic web service composition: A survey and problem formalization. **Computing and Informatics**, v. 30, n. 4, p. 793–827, 2011.

BENABOUD, R.; MAAMRI, R.; SAHNOUN, Z. Prefws3: Web services selection system based on semantics and user preferences. **Informatica (Slovenia)**, v. 40, n. 2, p. 257–274, 2016.

BERCHER, P.; MATTMÜLLER, R. Solving non-deterministic planning problems with pattern database heuristics. In: **32nd Annual German Conf. on AI, Paderborn, Germany**. [S.l.]: Springer, 2009. v. 5803, p. 57–64. ISBN 978-3-642-04616-2.

BERTOLI, P. **NuPDDL: nondeterminism and more in PDDL**. 2002. Acesso em: 29/07/2016.

BERTOLI, P. et al. **MBP: a Model Based Planner**. 2001.

BHATTACHARYA, A. et al. Hierarchical graph based approach for service composition. In: . [S.l.: s.n.], 2016. v. 2016-May, p. 1718–1722.

BIERE, A. et al. **Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications**. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2009. ISBN 1586039296, 9781586039295.

CABRI, N. S. nd G. **Adaptative, Dynimic and Resilient Systems**. [S.l.]: Taylor & Francis, 2014. (Mobile Services and Systems). ISBN 9797881439868485.

CARDOSO, J. **Semantic Web Services: Theory, Tools and Applications**. Hershey, PA, USA: IGI Global, 2007. ISBN 159904045X, 9781599040455.

CHAN, K. S. M. et al. A fault taxonomy for web service composition. In: _____. **Service-Oriented Computing - ICSOC 2007 Workshops: ICSOC 2007, International Workshops, Vienna, Austria, September 17, 2007, Revised Selected Papers**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 363–375. ISBN 978-3-540-93851-4.

CHEN, Y. et al. A robust service selection method based on uncertain qos. **Mathematical Problems in Engineering**, 2016. ISSN 1024-123X.

CHEN, Z. et al. A user dependent web service qos collaborative prediction approach using neighborhood regularized matrix factorization. In: **2016 IEEE 20th International Conference on Computer Supported Cooperative Work in Design (CSCWD)**. [S.l.: s.n.], 2016. p. 316–321.

CHRISTIAN, J.; BOHARA, M. Multi-agent web service composition using partially observable markov decision process. In: . [S.l.: s.n.], 2016.

CIMATTI, A. et al. Weak, strong, and strong cyclic planning via symbolic model checking. **Artificial Intelligence**, v. 147, p. 35 – 84, 2003. ISSN 0004-3702. Planning with Uncertainty and Incomplete Information.

CodeBrew Technologies. **ExpressBPEL Process Engine**. 2012. Disponível em: <http://codebrewtech.com>.

COPELAND, T. **Generating Parsers with JavaCC: An Easy-to-Use Guide tor Developers**. [S.l.]: Centennial Books, 2007. ISBN 0976221438.

COSTA, D. **DNS - Um Guia para Administradores de Redes**. [S.l.]: BRASPORT, 2007. ISBN 9788574522920.

COTE, E. M. de et al. Automated planning in repeated adversarial games. **CoRR**, abs/1203.3498, 2012.

DECHSUPA, C.; VATANAWOOD, W.; THONGTAK, A. Formal verification of web service orchestration using colored petri net. In: . [S.l.: s.n.], 2016. v. 1, p. 398–403.

ERNST, G. **GPS: A Case Study in Generality and Problem Solving**. [S.l.]: Academic Press, 1979.

ESTLIN, T. et al. Learning and planning for mars rover science. In: **In Proc. of IJCAI Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World Modeling, Planning, Learning, and Communicating**. [S.l.]: Morgan Kaufmann Publishers, 2003.

ETZIONI, O. et al. An approach to planning with incomplete information. In: **In Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning**. [S.l.]: Morgan Kaufmann, 1992. p. 115–125.

FENSEL, D.; KERRIGAN, M.; ZAREMBA, M. (Ed.). **Implementing Semantic Web Services: The SESA Framework**. Berlin: Springer, 2008. ISBN 978-3-540-77019-0.

FILIERI, A. et al. Discrete-time dynamic modeling for software and services composition as an extension of the markov chain approach. In: . [S.l.: s.n.], 2012. p. 557–562.

GEFFNER, H.; BONET, B. A concise introduction to models and methods for automated planning. **Synthesis Lectures on Artificial Intelligence and Machine Learning**, Morgan & Claypool Publishers, v. 8, n. 1, p. 1–141, 2013.

GHALLAB, M.; NAU, D.; TRAVERSO, P. **Automated Planning: Theory and Practice**. [S.l.]: Elsevier/Morgan Kaufmann Publishers, 2004. (The Morgan Kaufmann Series in Artificial Intelligence Series). ISBN 9781558608566.

GIACOMO, G. D.; PATRIZI, F. Automated composition of nondeterministic stateful services. In: LANEVE, C.; SU, J. (Ed.). **WS-FM**. [S.l.]: Springer, 2009. (Lecture Notes in Computer Science, v. 6194), p. 147–160. ISBN 978-3-642-14457-8.

GIUNCHIGLIA, E.; MARATEA, M. Sat-based planning with minimal-#actions plans and "soft"goals. In: **AI*IA 2007: Artificial Intelligence and Human-Oriented Computing, 10th Congress of the Italian Association for Artificial Intelligence, Rome, Italy, September 10-13, 2007, Proceedings**. [S.l.: s.n.], 2007. p. 422–433.

GREEN, C. Application of theorem proving to problem solving. In: . [S.l.]: Morgan Kaufmann, 1969. p. 219–239.

HART, P.; NILSSON, N.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. **Systems Science and Cybernetics, IEEE Transactions on**, v. 4, n. 2, p. 100–107, July 1968. ISSN 0536-1567.

HASAN, M. H.; JAAFAR, J.; HASSAN, M. F. Monitoring web services' quality of service: A literature review. **Artif. Intell. Rev.**, Kluwer Academic Publishers, v. 42, n. 4, p. 835–850, 2014. ISSN 0269-2821.

HELALI, R.; AZZOUNA, N. B.; GHEDIRA, K. Qos prediction in ubiquitous environments: An mlr based service selection approach. In: **2016 International Wireless Communications and Mobile Computing Conference (IWCMC)**. [S.l.: s.n.], 2016. p. 904–908.

HOFFMANN, J.; NEBEL, B. The ff planning system: Fast plan generation through heuristic search. **Journal Artificial Intelligence Research (JAIR)**, v. 14, p. 253–302, 2001.

HOFFMANN, J.; WEBER, I.; KRAFT, F. M. Sap speaks pddl: Exploiting a software-engineering model for planning in business process management. **Journal of Artificial Intelligence Research (JAIR)**, v. 44, p. 587–632, jul 2012.

JIMOH, F.; CHRPA, L.; MCCLUSKEY, T. The application of planning to urban traffic control. In: **24th International Conference on Automated Planning and Scheduling**. [S.l.: s.n.], 2014.

KALDELI, E.; LAZOVIK, A.; AIELLO, M. Domain-independent planning for services in uncertain and dynamic environments. **Artificial Intelligence**, v. 236, p. 30 – 64, 2016. ISSN 0004-3702.

KARDARAS, D. **Services Customization Using Web Technologies**. [S.l.]: Business Science Reference, 2012. (Premier reference source). ISBN 9781466616059.

KAUTZ, H. A.; SELMAN, B. Planning as satisfiability. In: **ECAI**. [S.l.: s.n.], 1992. p. 359–363.

KELLER, T.; EYERICH, P. A polynomial all outcome determinization for probabilistic planning. In: BACCHUS, F. et al. (Ed.). **ICAPS**. [S.l.]: AAAI, 2011.

KHOLY, M. E.; FATATRY, A. E. Frwsc: a framework for robust web service composition. **Service Oriented Computing and Applications**, p. 1–23, 2016. ISSN 1863-2394.

KOVACS, D. L. **Complete BNF description of PDDL 3.1**. [S.l.], 2011.

KUTER, U. et al. Using classical planners to solve nondeterministic planning problems. In: RINTANEN, J. et al. (Ed.). **ICAPS**. [S.l.]: AAAI, 2008. p. 190–197. ISBN 978-1-57735-386-7.

LAPRIE, J.-C. **From Dependability to Resilience**. [S.l.], 2008.

LEGG, S.; HUTTER, M. A collection of definitions of intelligence. In: **Proceedings of the 2007 Conference on Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms: Proceedings of the AGI Workshop 2006**. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2007. p. 17–24. ISBN 978-1-58603-758-1.

LEMOS, A. L.; DANIEL, F.; BENATALLAH, B. Web service composition: A survey of techniques and tools. **ACM Coumputing Surveys**, v. 48, n. 3, 2015.

LITTMAN, M. L. Probabilistic propositional planning: Representations and complexity. In: **Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)**. Menlo Park: AAAI Press, 1997. p. 748–754.

MA, Y. et al. A highly accurate prediction algorithm for unknown web service qos values. **IEEE Transactions on Services Computing**, v. 9, n. 4, p. 511–523, July 2016. ISSN 1939-1374.

MAKHORIN, A. **CNF Satisfiability Problem**. 2011.

MARKOU, G.; REFANIDIS, I. Cost-sensitive probabilistic contingent planning for web service composition. **International Journal on Artificial Intelligence Tools**, v. 25, n. 01, p. 1660001, 2016.

MARRELLA, A.; MECELLA, M.; SARDINA, S. Smartpm: An adaptive process management system through situation calculus, indigolog, and classical planning. 2014.

MATEO, J. et al. A coloured petri net approach to model and analyse stateful workflows based on ws-bpel and wsrf. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 8938, p. 389–404, 2015.

MCDERMOTT, D. et al. Pddl - the planning domain definition language. n. TR-98-003, 1998.

MEHDI, S.; ZAROUR, N. Composition of web services using multi agent based planning with high availability of web services. In: . [S.l.: s.n.], 2016. p. 10–15.

MI, C. et al. Reliability modeling and verification of bpel-based web services composition by probabilistic model checking. In: . [S.l.: s.n.], 2016. p. 149–154.

MUISE, C. J.; MCILRAITH, S. A.; BELLE, V. Non-deterministic planning with conditional effects. In: **Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014**. [S.l.: s.n.], 2014.

NIU, S. et al. Uclaoâˆ— and bhuc: Two novel planning algorithms for uncertain web service composition. In: . [S.l.: s.n.], 2016. p. 531–538.

ONAINDIA, E. et al. Simplanner: An execution-monitoring system for replanning in dynamic worlds. In: BRAZDIL, P.; JORGE, A. (Ed.). **EPIA**. [S.l.]: Springer, 2001. (Lecture Notes in Computer Science, v. 2258), p. 393–400. ISBN 3-540-43030-X.

OpenESB Community. **Open Enterprise Service Bus**. 2013. Disponível em: <http://www.open-esb.net>.

PAN, S. L.; MAO, Q. J. Semantic web service composition planner agent with a qos-aware selection model. In: **Web Information Systems and Mining, 2009. WISM 2009. International Conference on**. [S.l.: s.n.], 2009. p. 325–331.

PEER, J. Web service composition as ai planning - a survey. In: . University of St. Gallen, Switzerland: [s.n.], 2005.

QUINTERO, E. et al. Control of autonomous mobile robots with automated planning. **Journal of Physical Agents**, v. 5, n. 1, 2011.

RAO, J.; SU, X. A survey of automated web service composition methods. In: **Proceedings of the First International Conference on Semantic Web Services and Web Process Composition**. Berlin, Heidelberg: Springer-Verlag, 2005. (SWSWPC'04), p. 43–54. ISBN 3-540-24328-3, 978-3-540-24328-1.

RINTANEN, J. **Introduction to Automated Planning**. [S.l.], 2005.

SANDERSON, A. C.; MELLO, L. H. de; ZHANG, H. Assembly sequence planning. **AI Magazine**, v. 11, n. 1, p. 62–81, 1990.

SILVA, A. Sawczuk da; MA, H.; ZHANG, M. A graph-based qos-aware method for web service composition with branching. In: . [S.l.: s.n.], 2016. p. 131–132.

TINELLI, C. **Conversion to CNF**. [S.l.], 2010. Disponível em: <http://homepage.cs.uiowa.edu/ tinelli/classes/188/Fall10/notes/cnf-conversion.pdf>.

TODA, T.; SOH, T. Implementing efficient all solutions SAT solvers. **CoRR**, abs/1510.00523, 2015.

TRAVERSO, P.; PISTORE, M. Automated composition of semantic web services into executable processes. In: **The Semantic Web - ISWC 2004: Third International Semantic Web Conference,Hiroshima, Japan, November 7-11, 2004. Proceedings**. [S.l.: s.n.], 2004. p. 380–394.

VERVAEKE, J.; LILLICRAP, T. P.; RICHARDS, B. A. Relevance realization and the emerging framework in cognitive science. **J. Log. and Comput.**, Oxford University Press, Oxford, UK, v. 22, n. 1, p. 79–99, feb 2012. ISSN 0955-792X.

W3C. **Web Services Architecture**. [S.l.], February 2004. Disponível em: <http://www.w3.org/TR/ws-arch/wsa.pdf>.

WANG, H.; ZHENG, X. An online prediction approach for dynamic qos. In: **2016 IEEE International Conference on Services Computing (SCC)**. [S.l.: s.n.], 2016. p. 852–855.

WANG, P. et al. Automatic web service composition based on uncertainty execution effects. **Services Computing, IEEE Transactions on**, PP, n. 99, p. 1–1, 2015. ISSN 1939-1374.

WILKINSON, M. D.; VANDERVALK, B.; MCCARTHY, L. The semantic automated discovery and integration (sadi) web service design-pattern, api and reference implementation. **Journal of Biomedical Semantics**, v. 2, n. 1, p. 8, 2011. ISSN 2041-1480.

WINTERER, D.; MATTMULLER, R.; WEHRLE, M. Stubborn sets for fully observable nondeterministic planning. In: **Proceedings of the ICAPS-2015 Workshop on Model Checking and Automated Planning (MOCHAP 2015)**. [S.l.: s.n.], 2015. p. 6–12.

WU, C. et al. Qos prediction of web services based on two-phase k-means clustering. In: **Web Services (ICWS), 2015 IEEE International Conference on**. [S.l.: s.n.], 2015. p. 161–168.

YOUNES, H. L. S.; LITTMAN, M. L. **PPDDL 1.0: An extension to PDDL for expressing planning domains with probabilistic effects**. [S.l.], 2004.

ZHANG, R.-M.; LI, X.-B. Model checking of non-centralized automaton web service with amt bounded constraint. **International Journal of Multimedia and Ubiquitous Engineering**, v. 11, n. 3, p. 57–66, 2016.

ZHU, D. et al. An efficient dynamic programming algorithm for a new generalized lcs problem. **IAENG International Journal of Computer Science**, v. 43, n. 2, p. 204–211, 2016.

ZOU, G. et al. Dynamic composition of web services using efficient planners in large-scale service repository. **Knowledge-Based Systems**, v. 62, p. 98 – 112, 2014. ISSN 0950-7051.